

Paris 1971

T H E S E

présentée

A LA FACULTE DES SCIENCES
D'ORSAY

Pour obtenir le titre de Docteur Ingénieur

SPECIALITE : ELECTRONIQUE (TRAITEMENT DE L'INFORMATION)

par

Jean Claude CLAUDEL

Sujet de la Thèse : Conception d'un système de programmation
en langage symbolique

Soutenue le 2.4.1971 devant la Commission d'Examen

MM. PICINBONO	Président
DEBRAINE	Examineurs
DONADIEU	

T H E S E

présentée

A LA FACULTE DES SCIENCES

D'ORSAY

Pour obtenir le titre de Docteur Ingénieur

SPECIALITE : ELECTRONIQUE (TRAITEMENT DE L'INFORMATION)

par

Jean Claude CLAUDEL

Sujet de la Thèse : Conception d'un système de programmation
en langage symbolique

Soutenue le 2.4.1971 devant la Commission d'Examen

MM. PICINBONO

Président

DEBRAINE

DONADIEU

Examineurs

1888

1888

LA SOCIÉTÉ DES SCIENCES

DE BRUXELLES

Tout ce qui se rapporte à la Société des Sciences

de Bruxelles (Société des Sciences de Bruxelles)

1888

1888

La Société des Sciences de Bruxelles a l'honneur de vous adresser
le présent prospectus.



Leiden, le 10-12-1888

Leiden, le 10-12-1888

Leiden, le 10-12-1888

Ce travail a été réalisé au Laboratoire d'Electronique
Fondamentale - Faculté des Sciences d'Orsay.

Je tiens à exprimer mes sincères remerciements à Monsieur le
Professeur GRIVET et à Monsieur SAUZADE (Maître de Recherches) pour les
conditions de travail qu'ils m'ont accordées.

Je témoigne, à Monsieur P. DEBRAINE qui a dirigé mes travaux,
ma profonde gratitude et mon respectueux attachement.

Monsieur le Professeur PICINBONO a bien voulu, malgré ses
multiples occupations, accepter la présidence du Jury. Qu'il veuille
bien trouver ici l'expression de ma déférente gratitude.

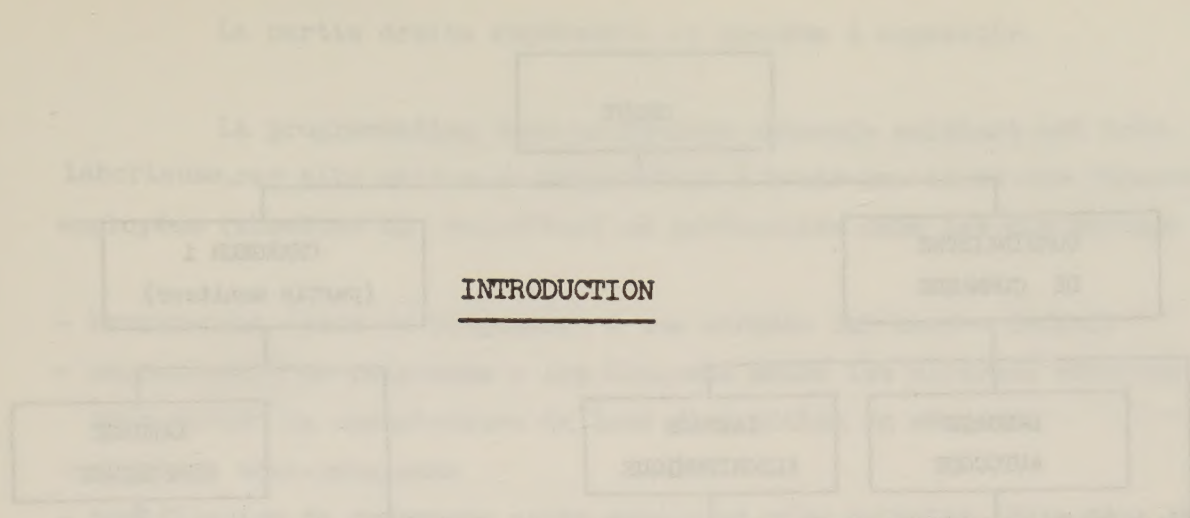
La présence dans ce jury de Monsieur le Professeur DONADIEU
m'est un honneur auquel je suis très sensible. Je le prie de croire en
ma respectueuse reconnaissance.

Je remercie également Madame COHEN et le personnel du
Service de Documentation de Saclay qui m'ont permis de présenter cette
thèse à la date prévue.

SOMMAIRE

	<u>Page</u>
INTRODUCTION	1
CH. 1 - CARACTERISTIQUES ESSENTIELLES DE LA MACHINE	4
I - REPRESENTATION DES INFORMATIONS	4
II - STRUCTURE D'ENSEMBLE	6
III - LANGAGE AUTOCODE CAB 500	7
IV - ASPECTS DES CARACTERISTIQUES INFLUANT PLUS PARTICULIEREMENT SUR LA PROGRAMMATION	12
CH. 2 - PROCESSUS ET LANGAGE D'ASSEMBLAGE	18
I - DEFINITION DU PROCESSUS D'ASSEMBLAGE	18
II - CLARIFICATION DES INSTRUCTIONS	20
III - SYNTAXE	23
IV - DIRECTIVES	29
V - DEFINITIONS ET REFERENCES EXTERNES	32
VI - EXPRESSIONS SYMBOLIQUES	35
CHA. 3 - ETUDE ET CONCEPTION DE L'ASSEMBLEUR	38
I - REMARQUES PRELIMINAIRES	38
II - ALGORITHME GENERAL DE L'ASSEMBLAGE	39
III - LECTURE DU PROGRAMME SOURCE	43
IV - CONFIGURATIONS ET GESTION DES TABLES	48
V - TRAITEMENT DES ETIQUETTES ET CODES OPERATION	61
VI - TRAITEMENT DES OPERANDES	75
VII - TRAITEMENT DES ERREURS	102
VIII - ASSEMBLAGE 2ème PHASE - EDITION DES LISTES	111
IX - DIRECTIVES	152
X - SOUS-PROGRAMMES	158
CHA. 4 - ETUDE DU CHARGEUR	169
I - TRAVAIL DU CHARGEUR	169
II - ORIENTATION DU TRAITEMENT	174
III - CHARGEMENT DES PROGRAMMES	176
CHA. 5 - MISE EN PLACE ET ESSAIS DU SYSTEME	217
I - OCCUPATION DE LA MEMOIRE CENTRALE	217
II - UTILISATION PRATIQUE DU SYSTEME	222
III - PROGRAMMES TESTS	223
IV - COMPARAISON AVEC L'ANCIEN SYSTEME	231
CONCLUSIONS	235

Digitized by the Internet Archive
in 2026



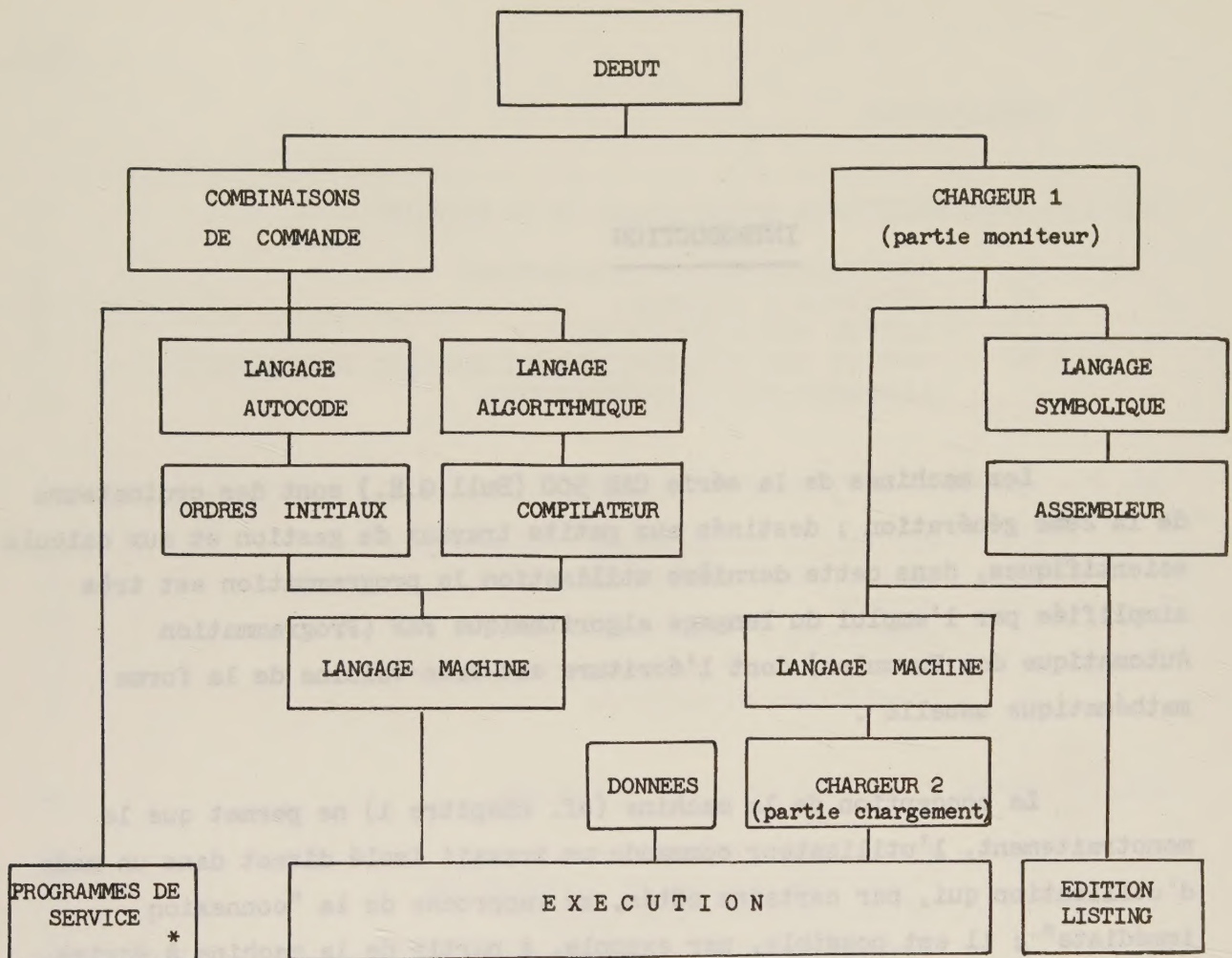
INTRODUCTION

Les machines de la série CAB 500 (Bull G.E.) sont des ordinateurs de la 2ème génération ; destinés aux petits travaux de gestion et aux calculs scientifiques, dans cette dernière utilisation la programmation est très simplifiée par l'emploi du langage algorithmique PAF (Programmation Automatique des Formules) dont l'écriture est très voisine de la forme mathématique usuelle .

La conception de la machine (cf. chapitre 1) ne permet que le monotraitement, l'utilisateur commande un travail isolé direct dans un mode d'utilisation qui, par certains côtés, se rapproche de la "connexion immédiate" ; il est possible, par exemple, à partir de la machine à écrire, d'interrompre un travail en cours d'exécution, de faire divers tests et même de modifier les instructions du programme (en autocode évidemment) et de reprendre le traitement au même endroit.

Le fonctionnement est orienté par les combinaisons de commande soit vers le PAF, soit vers le langage autocode et les programmes de service. Dans le cas de l'autocode le chargement s'effectue pas à pas à partir d'une adresse qui doit être fixée par le programmeur (translittération et chargement instruction par instruction).

Le système peut être représenté suivant la figure N° 1.



* Ils comprennent : le chargement de données. Edition, modification et mise au point de programmes

Figure 1

La partie droite représente le système à concevoir.

La programmation dans le langage autocode existant est très laborieuse, car elle oblige le programmeur à tenir une liste des adresses employées (absolues ou relatives) en particulier dans les cas suivant :

- branchement (saut de programme) à une adresse non encore définie
- segmentation de programme ; les liaisons entre les diverses sections nécessitent la connaissance de leur disposition en mémoire
- appel de sous-programme
- modification de programme après essais et mise au point. Elle mène généralement à l'adjonction ou à la suppression de certaines instructions et impose alors une réécriture complète du programme puisque la plupart des adresses sont modifiées.

Le modèle de base de la série est la CAB 500 A. Bien que les termes employés puissent avoir peu de signification pour le lecteur, nous donnerons ces caractéristiques dès maintenant, étant entendu que celles-ci seront étudiées par la suite :

- modèle 502 : extension capacité mémoire centrale
- modèle B : indice de saut généralisé
- modèle B1 : indice de saut et indexage généralisés et d'autres avantages jouant sur les entrées-sorties.

Le système a été essayé sur une CAB 502 B mais les programmes sont conçus de façon à être utilisables sur n'importe quel modèle, des simplifications intéressantes pourraient être apportées dans le cas du type B1, nous les signalerons également. La place en mémoire prise par l'ensemble est de 4 K (un peu plus de 3K pour l'assembleur et le reste pour le chargeur).

CHAPITRE 1

CARACTERISTIQUES ESSENTIELLES DE LA MACHINE

La CAB 500 est une machine à mots, mais les opérations élémentaires d'entrée-sortie doivent être programmées caractère par caractère.

I - REPRESENTATION DES INFORMATIONS

1 - La mémoire centrale est constituée par un tambour magnétique (temps d'accès moyen 10 ms) dont la capacité est de 16 ou 32 K mots (128 ou 256 pistes de 128 mémoires) de 33 positions binaires que l'on numérottera de 1 à 33 en partant de la droite (le digit b33 joue un rôle particulier de contrôle d'impairité). Dans le cas de la CAB 502B, les adresses seront donc repérées par un champ de 15 bits.

2 - La représentation des nombres peut s'effectuer soit en virgule fixée, soit en virgule flottante simple ou double longueur (les nombres négatifs sont exprimés dans la convention des compléments à 2^{31}).

a) - En virgule fixée, les opérations posant des problèmes de cadrage (multiplication et division) sont exécutées dans deux conventions prioritaires :

- entière : cadrage b1
- fractionnaire : cadrage b32

Le programmeur peut choisir une convention quelconque, mais il doit à chaque opération, suivre la position de la virgule et prévoir les impossibilités.

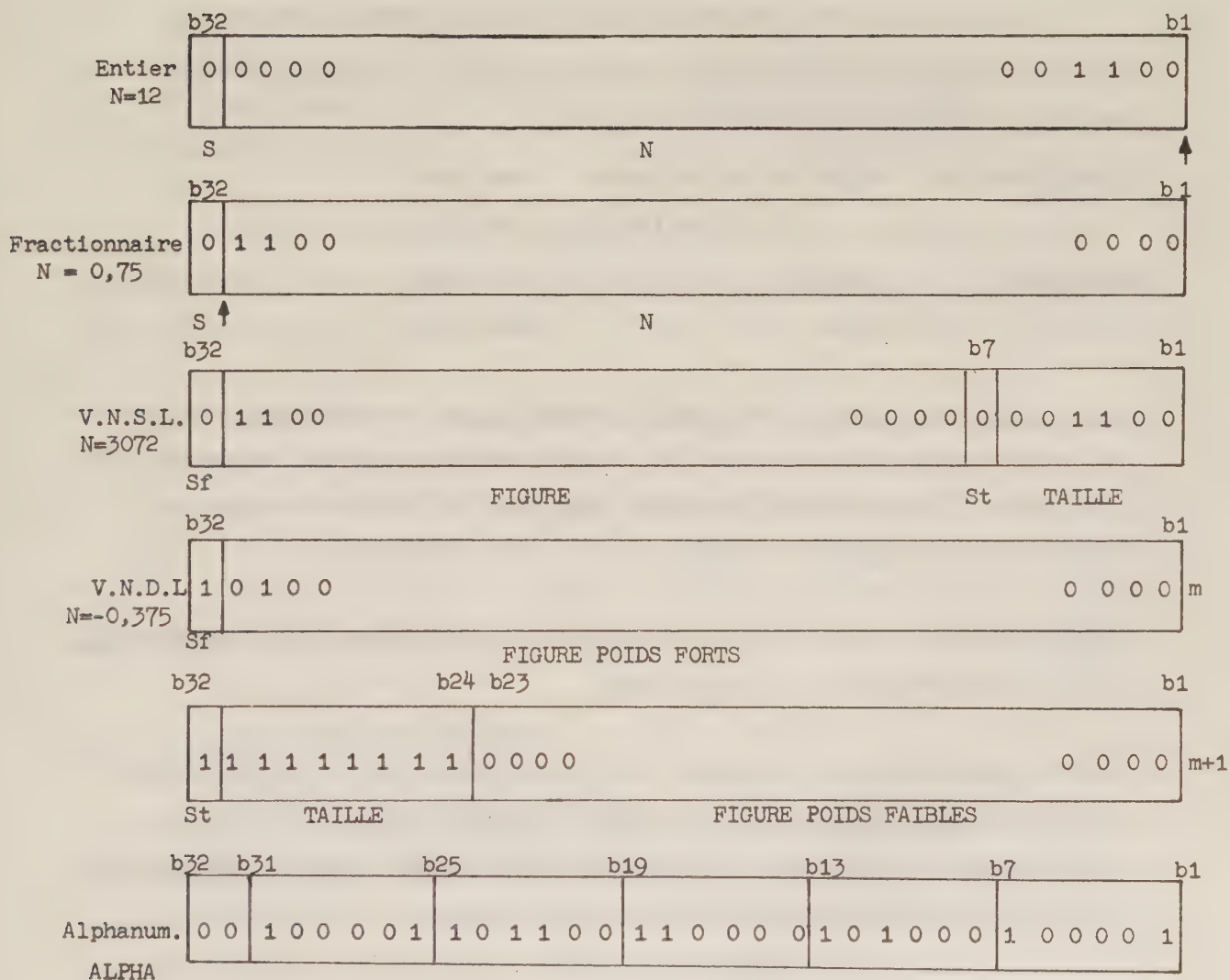
- b) - Les nombres en virgule flottante sont traduits dans la forme normalisée :

$$A = a.2^n$$

La figure est purement fractionnaire : $0,5 \leq |a| < 1$, la taille est entière et affectée d'un signe ; taille et figure sont exprimées dans les mêmes conventions que les nombres en virgule fixée.

3 - Les quantités alphanumériques sont représentées en DCB étendu à 6 digits : les nombres de 0 à 9, les lettres de 33 à 58, les caractères arithmétiques et spéciaux de 10 à 32 (cf. annexe n° 1).

4 - Configurations binaires



Nota : signification des abréviations

V.N.S.L. : virgule normalisée simple longueur

V.N.D.L. : virgule normalisée double longueur

II - STRUCTURE D'ENSEMBLE

1 - Les registres de travail sont au nombre de 16, numérotés de 0 à 15 ils contiennent chacun 33 positions binaires. En fait, la position 33 en registre est la répétition du digit de signe 32 et permet de contrôler le débordement de capacité.

Certains de ces registres généraux sont spécialisés :

- en multiplication et division en virgule fixée l'ensemble AC + MQ est réalisé par les registres 0 et 13 :

multiplication : signe et poids forts	→ (R0)
poids faibles	→ (R13)

division : quotient	→ (R0)
reste	→ (13)

- dans toutes les autres opérations arithmétiques l'accumulateur (AC) et le multiplicateur quotient (MQ) sont constitués par le registre spécifié dans l'instruction (en double longueur le résultat occupe ce registre et le registre suivant qui est alors implicite)
- l'accumulateur logique (ACL) est suivant spécification dans l'instruction l'un des deux registres : R13 ou R14
- le rôle particulier de registre d'index est tenu par le registre R15 ; dans le cas des modèles B1, l'indexage généralisé permet d'utiliser comme index n'importe quel registre général (défini dans l'instruction) moyennant une diminution de la zone d'adressage.

2 - Les organes d'entrée-sortie permettent deux types d'utilisation : machine à écrire (clavier) ou bandes perforées.

- a) - la flexowriter groupe, en plus du clavier, un lecteur mécanique et perforateur.
- b) - lecteur photoélectrique
- c) - perforateur rapide.

Pour mémoire nous donnons les performances de ces périphériques :

impression clavier :	10 caractères/seconde
perforation ou lecture :	10 caractères/seconde
lecture photoélectrique :	50 caractères/seconde
perforation "rapide" :	50 caractères/seconde

L'inconvénient majeur de ces périphériques (en dehors de leur nombre réduit) se situe au niveau programmation ; en effet l'utilisateur ne peut valider que manuellement le choix entre les diverses possibilités :

- lecteur photoélectrique ou lecteur mécanique
- perforateur rapide ou flexowriter
- impression et/ou perforation (sur la flexowriter).

Il n'y a que sur le modèle le plus évolué que les deux premiers choix sont programmables.

III - LANGAGE AUTOCODE CAB 500

Le but n'est pas, ici, de rapporter l'analyse détaillée des instructions telle qu'elle a du être faite ; nous nous bornerons à définir les caractéristiques essentielles du langage, réservant les nombreux cas particuliers à l'étude de l'assembleur.

Les instructions utilisables au nombre de 46, se décomposent en deux groupes :

- les instructions simples, câblées dans le bloc de calcul (13 instructions seulement), dont la durée est de 2 cycles mineurs (soit 310 micro secondes) ; elles comprennent les opérations arithmétiques simples, les opérations logiques et des opérations internes, telles que décalages, transferts, impression d'un caractère
- les instructions "microprogrammées" qui font appel à des opérateurs programmés en mémoire centrale. La durée de telles instructions peut aller de 40 millisecondes pour une addition en virgule flottante à 540 ms pour le calcul d'une exponentielle. Ces instructions sont :
 - des opérations arithmétiques en virgule flottante simple et double longueur
 - des opérations d'entrée-sortie dans les différentes représentations
 - des fonctions transcendantes usuelles.

1 - Structure générale des instructions

Il existe six champs possibles pour la rédaction d'une instruction ; ils sont dans l'ordre de leur écriture :

- a) - type opération (TO) : présent dans toute instruction
- b) - 1^{ère} adresse (AD1) : c'est l'adresse du 1^{er} opérande et éventuellement celui du résultat de l'opération. Il est constitué d'un numéro de registre : a
- c) - indice opératoire (IO) : est un caractère littéral qui apporte une information supplémentaire d'adresse ou modifie le résultat d'une instruction (un ordre d'addition avec indice E, fait précéder celle-ci d'une remise à zéro, l'instruction devient donc un chargement de registre)
- d) - 2^{ème} adresse (AD2) : désigne le deuxième opérande de l'instruction ; elle est composée d'un caractère littéral suivi d'une quantité numérique décimale :

- numéro de registre : R_b
- adresse mémoire : M_m
- quantités immédiates : V_v ou W_w

V_v si l'on utilise un entier V , tel que $0 \leq V < 2^{17}$

W_w si l'on utilise un entier n , tel que $n = W \cdot 2^{17}$ avec $0 \leq W < 2^{16}$

- caractère lu : $L\phi$, indique que le 2^{ème} opérande de certaines instructions sera un caractère lu par les organes d'entrée.

e) - caractère d'indexage : programmé par le signe + en fin d'instruction. (Digit b1 dans le mot machine).

f) - condition : certaines des instructions peuvent être conditionnelles ; le symbole littéral désignant le type de condition (CC) est suivi du numéro de registre (RC) sur lequel doit être effectué le test. Pour garder sa généralité, la condition doit être employée dans une instruction à une seule adresse (transfert de commande) sinon, seul le 2^{ème} opérande registre peut être utilisé (ces limitations sont dues à la capacité fixe du mot machine).

Les deux derniers groupes (condition et indexage) sont incompatibles, et cela réduit considérablement la puissance de la machine.

Il n'existe pas d'adressage indirect au sens général ; il est seulement présent dans les instructions possédant intrinsèquement ce caractère, c'est-à-dire les transferts de commande.

Exemples d'instructions CAB 500 :

A1R8 Addition du contenu du registre 8 $(R1) + (R8) \rightarrow (R1)$
 au contenu du registre 1

A12EV36 Chargement du registre 12 par la $36 \rightarrow (R12)$
 quantité immédiate : 36

RV267P5 Rupture vers l'instruction d'adresse $(R5) \geq 0 \ 267 \rightarrow (R1)$
 267 si le contenu de R5 est positif ou nul $(R5) < 0 \ (R1)+1 \rightarrow (R1)$

RM30	Rupture vers l'instruction dont l'adresse est contenue dans la mémoire d'adresse 30	$A (M30) \rightarrow (RI)$
ER6	Ecriture du caractère contenu dans R6	
X	Ineffectif	$(RI) + 1 \rightarrow (RI)$

Nota : on trouvera la liste des instructions CAB 500 en annexes n° 2 et n° 3

2 - Modèles d'instructions - Configurations binaires

Les configurations représentées ci-après correspondent aux instructions simples et à la majorité des instructions microprogrammées ; c'est dans cette dernière classe que l'on trouvera les cas particuliers. (Instructions d'entrée-sortie, calculs en double longueur).

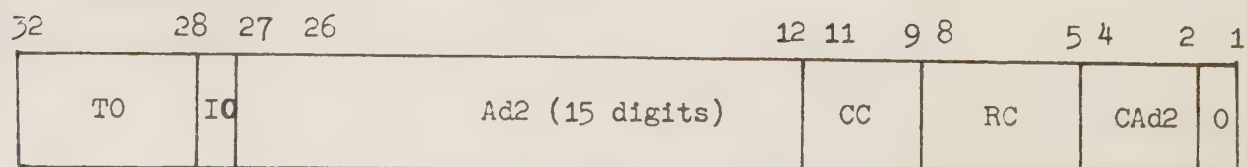
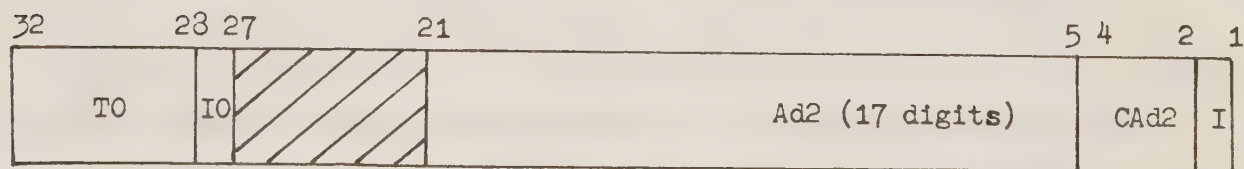
Il existe cinq modèles d'instructions répartis en trois classes :

Classe "0" : pas d'adresse d'opérande.

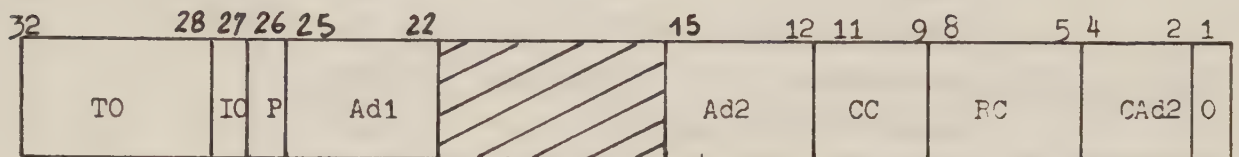
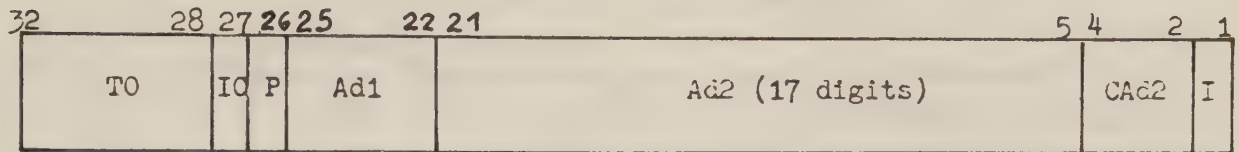
(par exemple instruction de non opération : X)



Classe "1" : instruction à une adresse d'opérande (2^{ème} adresse) ; la configuration change suivant la présence ou l'absence de condition.



Classe "2" : instruction à deux adresses d'opérandes (Ad1 et Ad2). Ici encore la condition modifie l'instruction en résuisant la zone d'adressage.



P est un digit d'appel de microprogramme (P = 0 pour une instruction câblée, 1 dans le cas contraire). Dans la suite de l'étude on désignera par "OP" (code opération) le champ des digits 32 à 26.

CC : code condition indiquant la nature du test à effectuer sur le registre condition explicite RC

CONDITION	SYMBOLE	CODE CONDITION	
		Décimal	Binaire
Inconditionnel	*	0	000
Variante (pupitre)	I *	1	001
(RC) $\neq$ 0	Y	2	010
(RC) = 0	Z	3	011
(RC) 0	N	4	100
(RC) 0	P	5	101
(RC) non débordé	B	6	110
(RC) débordé	D	7	111

* Pour ces deux cas le registre RC est ineffectif et peut être absent de l'instruction.

CAd2 : ce code de 3 digits permet d'indiquer les différents opérandes possibles

OPERANDE 2	CODE D'ADRESSE 2	
	Décimal	Binaire
Registre	0	000
Caractère lu	1	001
Mémoire	2	010
Mémoire et condition	3	011
Quantité V	4	100
Quantité V et condition	5	101
Quantité W	6	110

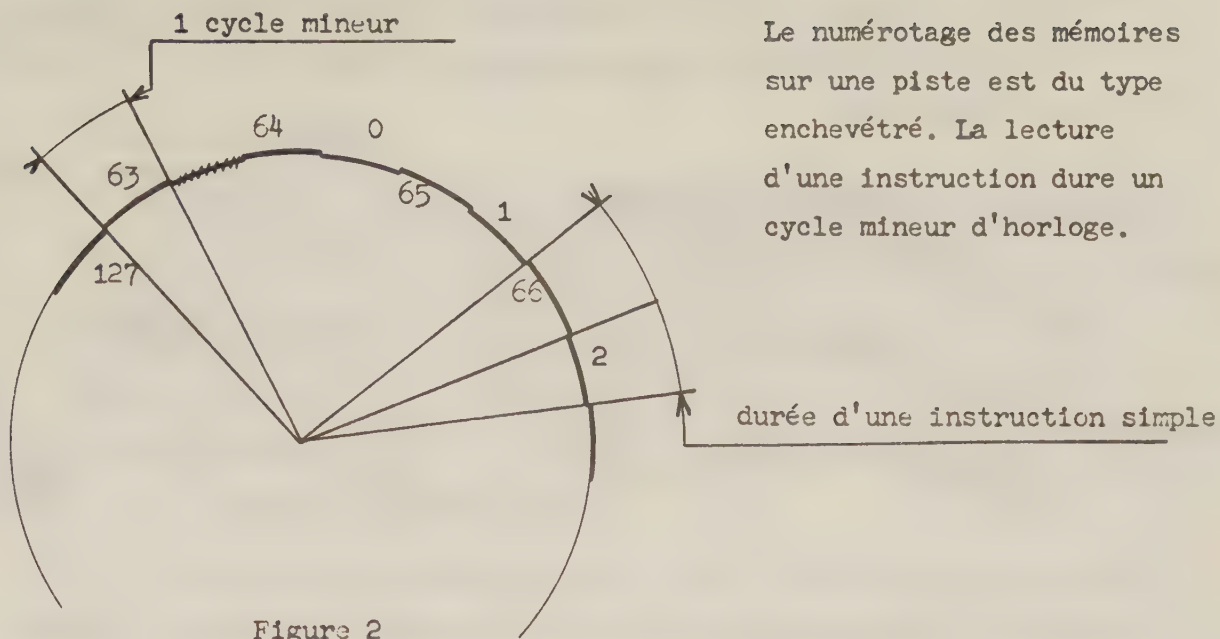
En examinant les configurations dans les différents modèles, on comprend facilement pourquoi dans une instruction conditionnelle à deux adresses, il n'est pas possible d'utiliser un deuxième opérande placé en mémoire ; la place dans le mot machine n'étant pas suffisante pour une adresse de 15 bits. L'adresse de type 1 étant toujours celle d'un registre, l'adresse 2 peut avoir au maximum 17 digits dans une instruction inconditionnelle, ceci explique la convention adoptées pour les quantités immédiates V et W au paragraphe III. 1. d.

IV - ASPECTS DES CARACTERISTIQUES INFLUANT PLUS PARTICULIEREMENT SUR LA PROGRAMMATION

1 - Mémoire centrale

La fonction de mémoire centrale remplie par un tambour magnétique pose un problème d'optimisation des adresses. L'étude faite dans ce paragraphe montre qu'il est très difficile sinon impossible de définir un procédé automatique d'optimisation satisfaisant.

a) - Disposition des mémoires (figure 2)

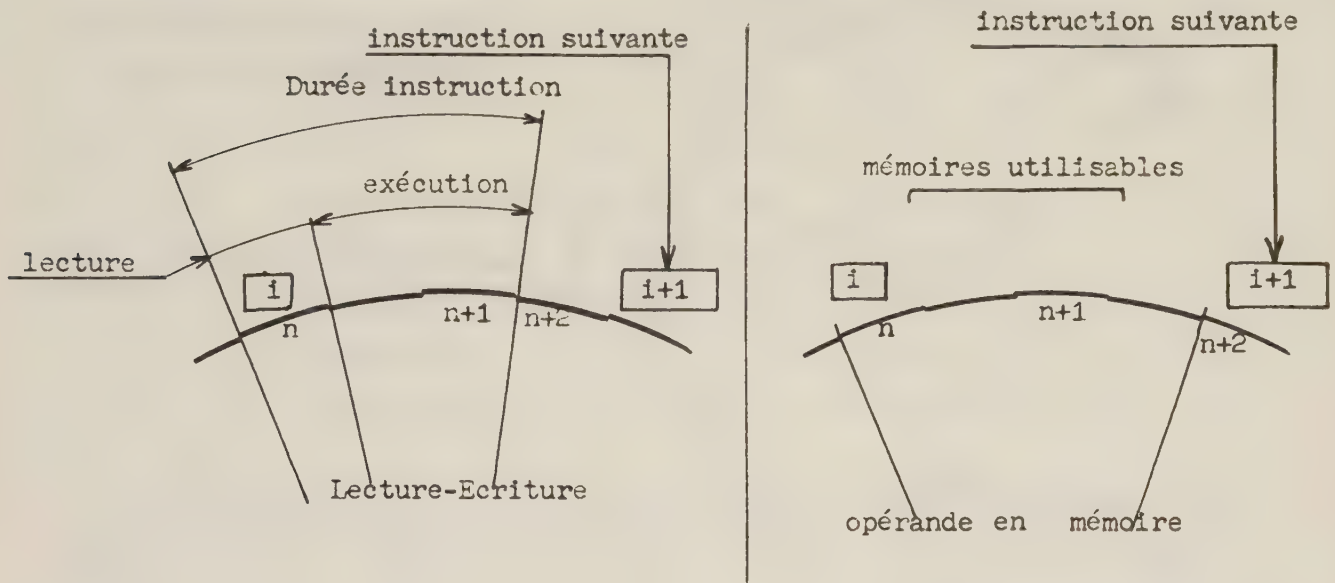


b) - Indice de saut :

Toute instruction câblée ne faisant intervenir, ni organe externe ni adresse mémoire nécessite deux cycles mineurs : le numérotage adopté permet de ne pas perdre systématiquement un tour de tambour (20 ms) pour trouver l'instruction normalement suivante .

Dans le cas d'une instruction faisant intervenir la lecture ou l'écriture d'un caractère alphanumérique, la durée totale est de 3 cycles mineurs ; de même un ordre utilisant un opérande situé dans une mémoire n , ne sera achevé qu'à la fin du passage de la mémoire $n + 1$ (lecture ou écriture dans la mémoire n et exécution).

Le rôle de l'indice de saut (programmé par S à la fin de l'instruction) permet de ne pas considérer la mémoire géographiquement à la suite en accordant 4 cycles mineurs au lieu de 2 pour l'exécution de l'opération.



Dans le cas d'une adresse mémoire, la perte de temps n'est évitée que si celle-ci est placée dans l'une des deux positions suivant l'instruction (soit $n + 1$, ou $n + 65$ modulo 128 dans le cas général).

Dans le cas des modèles B et B1, l'indice de saut généralisé supprime cette contrainte mais complique la programmation. L'instruction prise en charge par les organes centraux est toujours la plus proche de la mémoire utilisée dans l'instruction.

Le tableau annexe n° 4 donne l'adresse de l'instruction suivante en fonction de l'adresse de la mémoire utilisée.

Dans la pratique, l'emploi de l'indice de saut laisse des mémoires inutilisées qui, du point de vue chargement, représentent des trous qu'il faut combler, on placera dans celles-ci, soit des instructions de non opération, soit des constantes utilisées dans le programme.

L'optimisation est évidemment illusoire dans le cas d'une instruction microprogrammée puisque sa durée est fonction du type d'opération et des opérandes.

c) - Ruptures de séquence

On évitera une perte de temps systématique lors des débranchements (ruptures de séquence) en plaçant la première instruction de la nouvelle section en $n + 66$, si n est l'adresse de l'ordre de débranchement. Dans le cas d'un appel de sous-programme, l'optimisation ne peut généralement être faite, à la fois, sur l'entrée et la sortie.

2 - Organes d'entrées sorties

Il a été dit précédemment que le choix entre impression et perforation ne peut être programmé : cela apporte dans certains cas une complication importante du programme. Lors de l'édition de programmes assemblés, certaines parties sont à imprimer et perforer, d'autres à imprimer seulement; l'emploi d'instructions de sortie différentes permettrait de réaliser le travail dans la même section de programme.

3 - Indexage - adressage indirect et pointeurs

- a) - L'utilisation de plusieurs tableaux dans un même programme impose des mises en garde et des chargements fréquents du seul registre d'index utilisable.
- b) - L'absence d'adressage indirect apporte les mêmes inconvénients lorsque des registres ou des mémoires sont utilisés en "pointeurs" pour la gestion de tableaux.

Dans l'élaboration des programmes constituant le système nous ne reviendrons pas sur ces aspects particuliers de la programmation, étant entendu qu'ils ne seront pas négligés et que les programmes seront

rédigés en conséquence, par exemple, afin de minimiser le nombre d'appels tambour, nous ferons une utilisation maximale des 16 registres généraux disponibles, l'annexe n° 11 donne les tableaux d'utilisation de ces registres au cours des différentes phases de l'assemblage et du chargement.

Remarque : dans le cas d'un travail en boucle comportant peu d'instructions, il peut être intéressant de répéter la séquence de programme, ce qui allonge l'écriture mais diminue considérablement le temps d'exécution. En effet le retour en début de boucle implique l'attente d'une révolution du tambour ; si t est le temps d'exécution d'une instruction, T le temps de révolution du tambour, M le nombre de séquences identiques à répéter et N le nombre d'instructions de la séquence, les temps nécessaires dans chaque cas sont :

Temps dans le cas de boucles : $T_b = N.t + (M - 1) T$

Temps dans le cas de répétition de séquence : $T_s = M.N.t$

Si l'on étudie le rapport $R = T_b/T_s$ en fonction de M et N , on obtient le réseau représenté par la figure 3

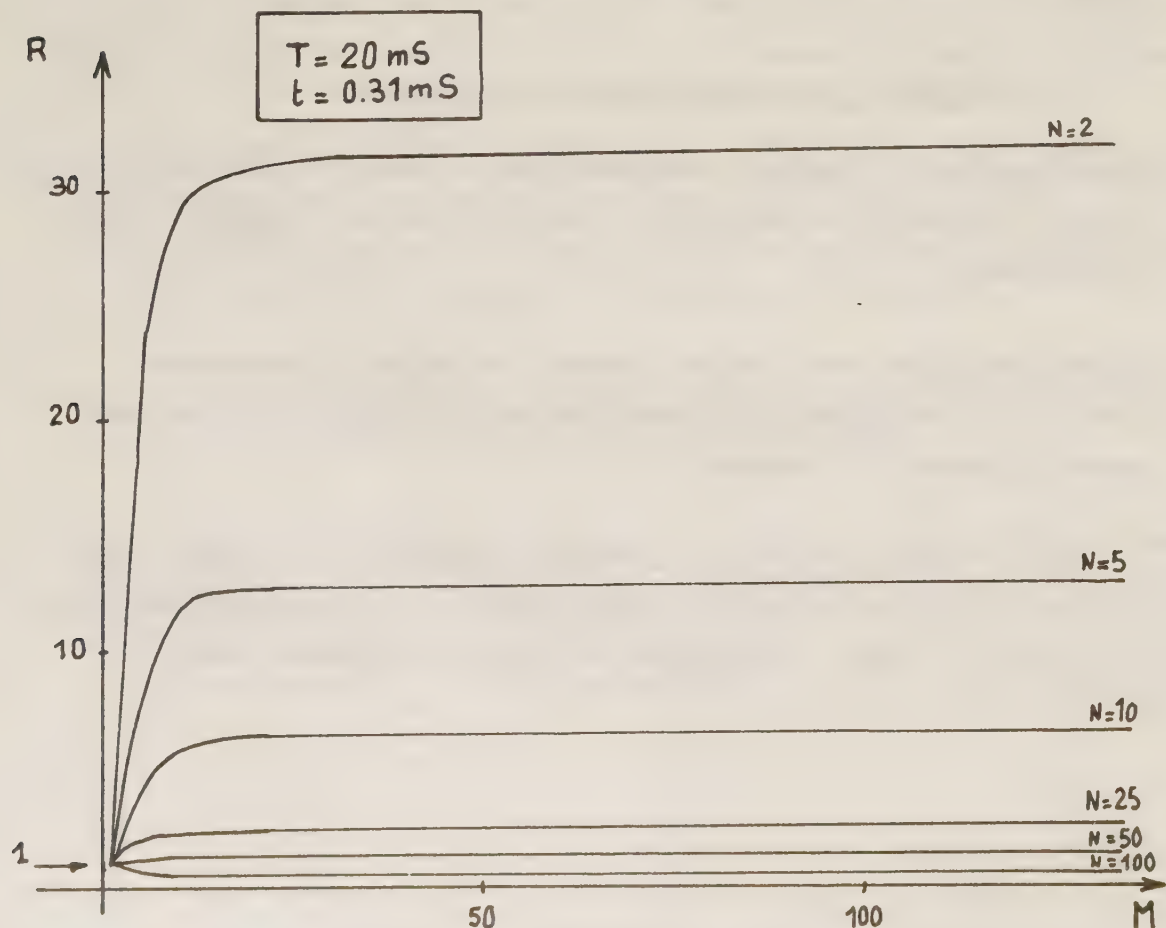


Figure 3

Le gain de temps est appréciable si le nombre d'instructions N n'est pas trop important et le nombre de boucles suffisant. Toutefois on ne peut aller bien loin, car la répétition de séquences identiques devient vite fastidieuse et multiplie les risques d'erreurs.

Exemple : Soit à lire un groupement de 6 caractères au maximum, par une séquence utilisant 10 instructions :

1er cas :

10 instructions à écrire.

Temps d'exécution : $T_b = 10.0,31 + 5.20 = \underline{103 \text{ ms}}$

2ème cas :

60 instructions à écrire.

Temps d'exécution : $T_s = 6.10.0,31 = \underline{19 \text{ ms}}$

CHAPITRE 2

PROCESSUS ET LANGAGE D'ASSEMBLAGE

Le langage symbolique que nous allons définir, doit donner à l'utilisateur un outil de programmation suffisamment puissant tout en restant compatible avec les possibilités de la machine (temps d'exécution, place en mémoire). Afin de faciliter la programmation, nous donnons au langage symbolique une structure aussi simple que possible en évitant les trop nombreux cas particuliers et les redondances du langage autocodé CAB 500 ; nous cherchons, sans toutefois dénaturer le système, un langage assez proche de ceux existant pour des ordinateurs plus modernes.

I - DEFINITION DU PROCESSUS D'ASSEMBLAGE

1 - Assemblage en un passage

Un assembleur utilise généralement plusieurs fois le programme source ; au premier passage le système construit la "table des symboles" en affectant une adresse aux étiquettes et aux symboles correspondants. Le second passage s'impose parce que la majorité des programmes utilisent des symboles qui ne sont pas encore définis lors de leur emploi dans la section de programmes à assembler.

A la fin de la première passe, l'assembleur connaît normalement toutes les adresses utilisées, il peut alors générer le programme objet lors d'une deuxième lecture du programme source ou compléter une version semi-codée au premier passage.

Le mode de travail est intéressant quand les périphériques sont suffisamment rapides et pratiques, sinon il est préférable d'éviter les opérations d'entrée-sortie.

C'est le cas de notre étude : le programme ne peut être enregistré qu'en mémoire centrale à partir du clavier de la machine à écrire ou d'une bande perforée.

Dès maintenant, nous pouvons donc prévoir que l'assemblage sera fait en un seul passage, le programme source étant stocké en mémoire afin d'être disponible immédiatement pour l'impression du listing objet.

2 - Problème des adresses

La traduction d'une instruction symbolique doit être différée dans trois cas :

- Instruction de calcul qui utilise un symbole représentant une constante ou une variable dont l'emplacement n'est pas encore définie lors de sa rencontre dans le programme.
 - Instruction de branchement à une adresse située dans la suite du programme.
 - Référence à une adresse externe à la section assemblée (appel de sous-programme - segmentation de programmes longs.....).
- a) - La première difficulté est éliminée si l'on admet que la définition des données doit précéder le programme principal. Cette règle n'est pas très contraignante pour le programmeur, car elle n'intervient en fait que lors de la frappe du programme.
- b) - Le deuxième problème peut être résolu de deux façons :
- affectation systématique d'une nouvelle zone de mémoire à chaque nouveau symbole non défini. Cette solution implique une gestion importante de la mémoire surtout si l'on veut éviter les pertes de place.
 - l'emploi de la méthode de la liste (chaînage des adresses) permet de résoudre plus facilement cette question.

Quand l'assembleur rencontre un symbole non défini, il le place dans la table des symboles avec l'adresse de l'instruction qui l'utilise (valeur actuelle du compteur d'adresse) et il continue l'exploration du programme ; chaque fois que ce symbole est cité dans une instruction, le contenu du compteur d'adresse remplace dans la table des symboles l'adresse de la précédente utilisation qui est préalablement transférée dans la zone d'adressage de cette nouvelle instruction. Lors de la découverte de l'étiquette correspondante, l'assembleur charge l'adresse ainsi définie dans chaque instruction incomplète. Le dernier élément de la chaîne est indiquée par la table des symboles et chaque instruction contient l'adresse de la précédente jusqu'à la première instruction dont le champ d'adresse est nul.

c) - Le troisième cas ne peut être réglé lors de l'assemblage puisqu'il s'agit de liaisons entre programmes assemblés séparément. Le travail est alors réalisable avec la même méthode, mais cette fois, par le chargeur.

Ces questions seront examinées en détail dans la suite de l'exposé.

Nota : la méthode de la liste pourrait être employée pour résoudre la première difficulté, mais cela alourdirait considérablement l'assemblage.

II - CLARIFICATION DES INSTRUCTIONS

Afin de clarifier le langage symbolique et d'alléger le travail d'assemblage et de chargement, il est possible de réduire le nombre de classes d'instructions (cf. chapitre précédent paragraphe III-2) en n'affectant que peu les possibilités de la machine.

La simplification consiste à ne garder la conditionnalité que sur les instructions de branchement.

1 - Incidence sur la programmation

- a) - les instructions sans opérande sont l'arrêt de la machine et la non opération. Cette dernière instruction ne se programmant jamais avec une condition, seule l'instruction d'arrêt est affectée.
- b) - les instructions comportant un seul opérande sont les instructions de branchement et d'écriture d'un caractère ; là encore, la restriction ne joue que sur une instruction et les possibilités d'adressage des instructions de branchement ne sont pas réduites par la diminution du champ d'adresse de 17 à 15 digits. En effet, l'adresse désignée est au maximum égale à 32767 (capacité mémoire maximale : 32 K), elle se traduit donc par 15 digits au plus.
- c) - une instruction à deux opérandes ne pouvait être conditionnelle que si l'adresse dite de type 2 était celle d'un registre ; la restriction par rapport au langage initial, affecte, ici encore, un pourcentage réduit des instructions. (voir annexe n° 6. Utilisation des instructions). Il faut remarquer par ailleurs, que le fait de supprimer cette possibilité de condition sur certaines instructions ne complique pas la programmation, elle impose simplement d'ajouter une instruction de branchement conditionnelle à celle qui ne peut plus l'être :

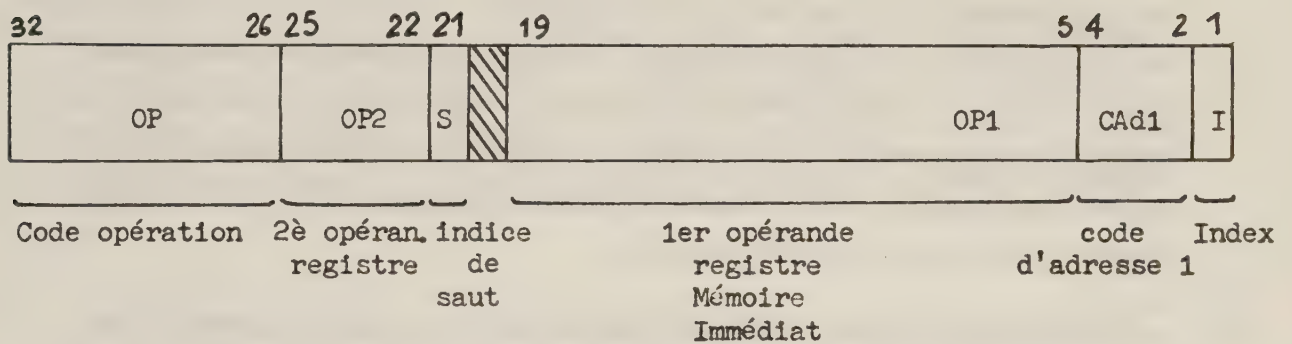
Exemple : si dans un programme rédigé dans l'autocode CAB 500, l'ordre n était un ordre conditionnel : A 2 R 8 P 5

(addition du contenu du registre 8 au contenu du registre 2 si le contenu du registre 5 est positif ou nul). Dans le nouveau langage l'ordre n sera une instruction de branchement conditionnel à l'instruction d'adresse (n + 2), l'ordre situé en (n + 1) étant l'instruction d'addition du registre 8 au registre 2.

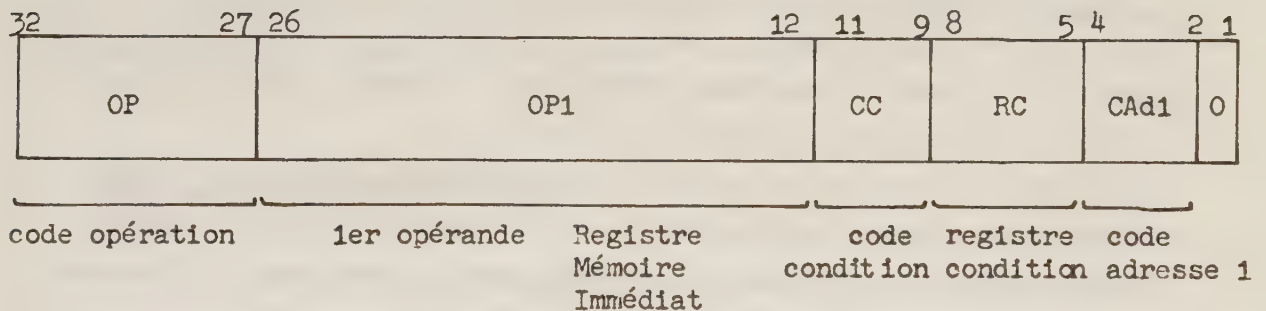
2 - Configurations binaires

La convention adoptée permet de définir deux types d'instructions binaires :

a) - instructions normales : de zéro à deux opérandes



b) - instructions conditionnelles (branchements) :



III - SYNTAXE

Le programme sera constitué d'un certain nombre de lignes symboliques, la dernière étant obligatoirement une directive "FIN".

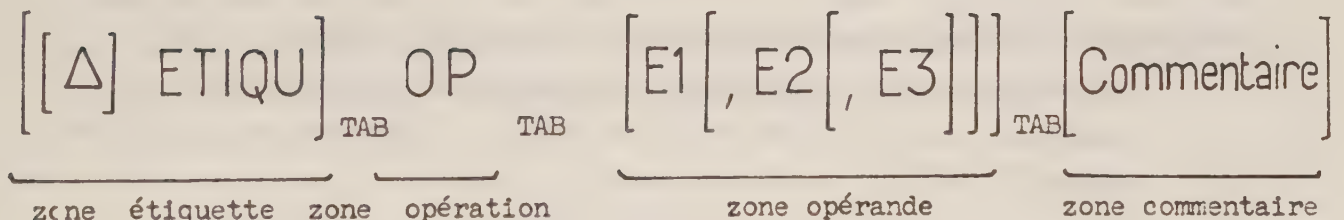
1 - Ligne symbolique

La ligne est l'unité de traitement de l'assembleur, la fin de ligne est indiquée par le caractère retour de chariot de la machine à écrire (RC). Elle constitue une unité logique décomposée en quatre zones : zone étiquette, zone opération, zone opérande et zone commentaires.

Le caractère de séparation des zones est la tabulation (TAB), ceci permet d'avoir une présentation du programme en colonnes même si la longueur des zones est variable.

La ligne est constituée par une instruction, une directive (pseudo-instruction pour l'assembleur) et par un éventuel commentaire.

2 - Composition d'une instruction



Les symboles entre crochets ne sont pas obligatoires dans une instruction.

△ Ce symbole indique que l'étiquette qui suit est une définition externe c'est-à-dire qu'elle constitue une entrée secondaire dans le programme.

ETIQU Nom symbolique constitué de cinq caractères alphanumériques au maximum, dont le premier est obligatoirement une lettre.

OP Code opération, toujours présent dans une instruction.

E1 Expression désignant le premier opérande

- numéro de registre
- expression symbolique
- facteur immédiat (entier décimal ou hexadécimal)

E2 Nombre décimal caractérisant

- numéro de registre (instructions à 2 opérandes)
- facteur de cadrage (instruction d'impression)
- code condition (instruction de branchement).

E3 Nombre décimal ou symbole

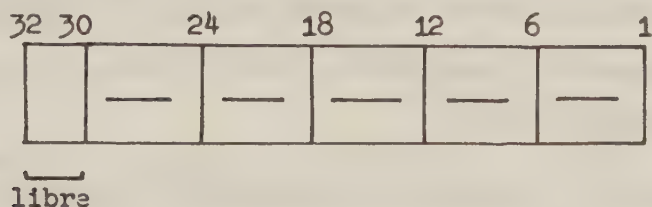
- facteur de cadrage (impression)
- numéro de registre condition (branchement)
- modificateur + (indexage)
- S (indice de saut).

COMMENTAIRE : La zone commentaire permet de placer des repères et des annotations dans le programme ; celle-ci est ignorée par l'assembleur, mais elle est restituée lors de l'impression de la liste objet.

Voyons maintenant, plus en détail, ces diverses parties d'une ligne de programme.

3 - Zone étiquette

Le nombre de caractères alphanumériques (lettres et chiffres à l'exclusion de tout autre caractère) est limité à cinq par la capacité même du mot machine. Un caractère est représenté par six digits binaires et le mot contient trente deux bits.

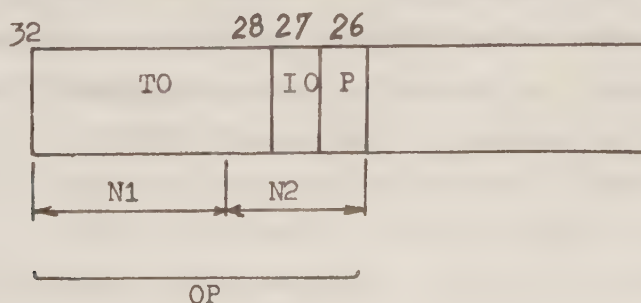


Les digits 32 et 31 seront toujours libres pour une mémoire contenant un symbole et pourront être utilisés par le système pour des informations complémentaires.

Une étiquette placée à la première ligne du programme (compteur d'adresse nul) définit le nom symbolique du programme. L'absence d'étiquette est indiquée par le caractère tabulation en début de ligne.

4 - Codes opération (annexe 2 et 3)

L'utilisation de l'indice opératoire IO (cf. langage autocode CAB 500) et de microprogrammes se comportant du point de vue programmation comme des instructions câblées, porte à 68 le nombre d'opérations possibles. Nous convenons de regrouper sous un même code opération (OP) désigné par un nombre hexadécimal, l'ensemble des digits 26 à 32 du mot machine.



Le code opération ne comportant que sept digits, le dernier digit du chiffre hexadécimal N2 sera toujours nul, et N2 sera donc toujours pair.

Nous avons vu précédemment, que chaque opération peut faire intervenir plusieurs types de 1^{er} opérande suivant les instructions. Dans le langage symbolique, ce renseignement complémentaire sera regroupé avec le code opération puisqu'il change la nature de l'ordre indiqué :

R indique un 1^{er} opérande en registre

I indique un 1^{er} opérande immédiat

L'absence de R et I indique implicitement un opérande en mémoire.

Il n'y a pas de confusion possible si aucun des codes opération, dans sa forme réduite, ne se termine par R ou I, ce qui est le cas.

Exemple

A	addition	}	1 seul code opération <u>A</u> et un code désignant la nature du 1 ^{er} opérande.
AR	addition registre		
AI	addition immédiate		

5 - Zone opérande

a) - 1^{er} opérande

L'assemblage est compliqué par le fait que suivant les instructions, les possibilités d'adressage du 1^{er} opérande sont différentes dans leur forme et dans leurs limites.

Dans l'ordre des possibilités on trouve en 1^{er} opérande :
registre seul : instructions d'entrée (câblées ou programmées)
registre et mémoire : la plupart des microprogrammes
registre mémoire et immédiat : instructions câblées.

Suivant les instructions le champ d'utilisation des valeurs immédiates ou des registres est variable. Par exemple, les instructions microprogrammées utilisent dans leur majorité les registres 9 à 14 pour l'élaboration du résultat, interdisant ainsi l'emploi de ces registres comme contenu des opérandes.

L'annexe n° 3 fournit la liste des instructions par classe avec les limites et restrictions correspondantes.

- le numéro de registre est exprimé par un nombre décimal compris entre 0 et 15 (sauf restriction)
- l'adresse désignant une mémoire, n'est limitée que par la capacité de la mémoire centrale
- les quantités immédiates sont désignées par un entier décimal ou hexadécimal. Une quantité hexadécimale est précédée des symboles X =
Le symbole $\times$ (multiplication) représente le facteur d'échelle : 2^{17}
correspondant aux quantités immédiates W du langage autocode CAB 500.
(chapitre précédent paragraphe III 1 d).

Exemples de quantités immédiates utilisables dans les instructions symboliques :

35	valeur décimale	35
X = 2A	valeur décimale	42
$\times$ 2	valeur décimale	262 144 (2.2^{17})
$\times$ X = 20	valeur décimale	4 194 304 (32.2^{17})

b) - Second opérande

- l'utilisation du registre est soumis aux mêmes conditions que pour le premier opérande
- facteur de cadrage :
de 0 à 15 (décimal) dans les opérations d'impression de nombres
représentés dans la convention de la virgule fixée
de 0 à 31 pour la convention de la virgule normalisée
- code condition :
de 0 à 7 (cf. tableau paragraphe III 2 chapitre précédent)

c) - Troisième opérande

- le facteur de cadrage possède les mêmes limites qu'en deuxième opérande
- le registre condition est exprimé, sans limitation particulière, par un nombre décimal de 0 à 15

6 - Zone commentaire

Aucune restriction n'est imposée dans la rédaction des commentaires.

Toute ligne de programme débutant par le symbole / (barre de fraction) est considérée comme un commentaire et ne donne lieu à aucune génération d'instruction. (L'utilisation de ce symbole en cours de commentaire est évidemment possible et n'a d'autre signification que celle donnée par le programmeur)

La fin de ligne étant obligatoirement un retour de chariot (RC), tout commentaire devant se poursuivre sur une ou plusieurs lignes est précédé, à chaque fois, du symbole /.

Une ligne sans instruction (deux retours de chariot consécutifs) est traitée comme un commentaire, (le compteur d'adresse n'évolue pas) ; il n'est donc pas nécessaire d'utiliser la barre de fraction pour indiquer un saut de ligne.

7 - Exemples

Etiquette	Opération	Opérandes	Commentaire
TEST 1	AR	1,15	ADDITION REGISTRE 1 au REGISTRE 15
ALPHA	CI	64,3	Chargement immédiat "64" au registre 3
	BCI	JUMP1,6,13	Branchement conditionnel à l'adresse symbolique JUMP1 si R13 non débordé
Δ BETA	CC	SIN, Ø, +	Chargement complémenté indexé en registre Ø
	HSE		Halte et séquence.

IV - DIRECTIVES

Les directives sont les instructions destinées à l'assembleur, pour l'instant nous ne définissons que les directives strictement nécessaires à l'écriture des programmes. Elles appartiennent à deux catégories:

1 - Définition de données

- "DFC" : définition de constante

zone étiquette	opération	opérande	commentaire
$[\Delta] \text{ETIQU}$	DFC	$[\underline{N}] \text{ T } \underline{C} \text{ } [E]$	$[\text{COMMENT}]$

N est un facteur d'itération différent de zéro, sa présence indique que la constante définie est à répéter N fois.

T détermine le type de constante :

F = Décimal avec ou sans signe

X = Entier hexadécimal avec ou sans signe

S = Décimal normalisé simple longueur

D = Décimal normalisé double longueur

C est la constante elle-même, elle peut être suivie d'une modification d'échelle

En est le modificateur d'échelle

/ opérateur d'échelle binaire pour F et X ; n est entier et compris entre -32 et +32

Δ opérateur d'échelle décimale pour S et D ; n, entier, n'est limité que par les valeurs minimales et maximales représentables dans la convention de la virgule normalisée.

Ainsi la suite des instructions :

CONST	DFC	F=-12	<u>FIXE</u>
	DFC	2F=6,75/5	<u>FIXE</u>
	DFC	X=FE27000/-7	FIXE <u>HEXADECIMAL</u>
	DFC	3S=0,5 Δ 4	FLOTTANT <u>SIMPLE</u>
	DFC	D=378 Δ -2	FLOTTANT <u>DOUBLE</u>

génère à partir de l'adresse ayant comme symbole CONST, les mots machine suivants (hexadécimal).

CONST	FFFFFFF4
+1	000000D8
+2	000000D8
+3	001FC4E0
+4	4E20000D
+5	4E20000D
+6	4E20000D
+7	78F5C28F
+8	012E147B

- "TEXTE" : génération de texte

Zone étiquette	opération	opérande
$\left[\left[\Delta \right] \text{ETIQU} \right]$	TEXTE	E

E est une suite de caractères alphanumériques quelconque, la fin de l'expression est donnée par la fin de ligne : présence d'un retour de chariot. L'expression E peut contenir tous les caractères, en particulier des tabulations, cette directive ne peut donc pas comporter une zone de commentaires.

L'expression est décomposée en groupe de cinq caractères dont le premier est cadré en b25, ces groupes sont transférés dans les mots-machine

successifs, le dernier mot étant éventuellement complété par des codes "espace" de la machine à écrire (en code interne : $(15)_{10}$).

La suite des instructions :

TEXTE	TEST	DIRECTIVE
TEXTE	STOP	

génère les mots suivants : (représentation hexadécimale)

34973DOF
24A72963
34A7694F
33D2FCOF

2 - Pseudo-opérations

- "RES" : réservation de bloc

cette directive permet de réserver des positions de mémoire.

Etiquette	Opération	Opérande	Commentaire
$\left[\left[\Delta \right] \text{ETIQU} \right]$	RES	<u>N</u>	$\left[\text{COMMENT} \right]$

N désigne le nombre de positions à réserver à partir de la valeur actuelle du compteur d'adresse.

- "FIN" : fin de programme ou de macro-instruction

Etiquette	Opération	Opérande	Commentaire
	FIN	E	$\left[\text{COMMENT} \right]$

Cette instruction indique à l'assembleur que la section de programme est terminée ou qu'il s'agit de la fin d'une macro-instruction.

Quand la zone opérande existe, elle indique l'adresse de début (exécution) du programme.

L'opérande E peut être une constante ou un symbole défini par une étiquette au cours du programme.

Cette pseudo-opération ne correspond à aucune instruction du point de vue de la section de programme assemblée et ne doit donc pas être précédée d'une étiquette. (Un branchement ou une référence à cette pseudo-opération n'ayant aucun sens).

Nous ne définissons pas pour l'instant de directives fixant l'origine de chargement du programme ; nous admettons que le système fixe lui-même cette adresse et que les programmes sont toujours assemblés en adresses relatives.

V - DEFINITIONS ET REFERENCES EXTERNES

Lorsqu'un programme principal utilise des sous-programmes indépendants ou lorsqu'un programme est trop volumineux pour être assemblé en un seul segment, on est conduit à assembler séparément des programmes interdépendants. Les liaisons externes sont assurées par l'emploi des définitions et références externes.

1 - Définitions externes

Une définition symbolique (par une étiquette) est dite externe lorsqu'elle est précédée du symbole delta (Δ), elle indique qu'il est fait référence à ce symbole dans un ou plusieurs segments externes.

2 - Références externes

Dans un programme utilisé seul, les étiquettes sont internes c'est-à-dire définies à l'intérieur du programme. Quand un symbole ne peut être défini à l'intérieur du segment à assembler, les références à ce

symbole sont appelées "références externes". Celles-ci n'apparaissent que dans la zone d'adresse symbolique des instructions et doivent s'y trouver seules. (Nous en verrons la raison au paragraphe VI. 3).

L'assembleur crée une référence externe dans deux cas :

a) - instructions de branchement

Elles peuvent faire appel à un symbole défini ultérieurement ; ce n'est donc qu'après la rencontre de la directive "FIN", que l'assembleur sait si le symbole a été effectivement défini dans le programme. Tous les symboles qui ne sont pas encore définis sont donc des références externes.

b) - instructions quelconques

Nous avons vu qu'elle ne doivent utiliser que des symboles définis (variables et constantes en début de programme) ; si le symbole employé par l'instruction n'est pas encore défini, l'assembleur le classe systématiquement dans les "références externes". Une définition ultérieure de ce symbole constituerait une faute de syntaxe et le système générerait un diagnostic de multidéfinition.

3 - Listing et chargement

Les liaisons entre les divers segments ou sous-programmes ne sont connues qu'au moment du chargement ; pour transmettre au chargeur les renseignements nécessaire à la réalisation de ces liaisons, l'assembleur génère à la fin du programme binaire objet une table des définitions externes, et une table des références externes. Celles-ci sont aussi présentes dans la liste des programmes source et objet.

Les références externes faites à un même symbole constituent une liste enchaînée, c'est-à-dire que chaque symbole de la table des références externes est en correspondance avec l'adresse binaire du dernier

emplacement contenant cette référence tandis que la partie adresse de l'instruction incomplète donne l'adresse du dernier emplacement utilisant une référence externe qui correspond à ce symbole ; la liste se termine quand la partie adresse de l'emplacement est nulle.

Le programme ci-après, assemblé par le système, permet de bien comprendre la méthode de liaison : la lecture des références externes donne le dernier emplacement correspondant à chaque référence ; en parcourant le listing en sens inverse l'utilisateur retrouve toutes les instructions qui se réfèrent au même symbole.

La colonne ADR1, donnant l'adresse mémoire contenue dans l'instruction binaire, il est facile de reconstituer les deux chaînes :

```

COSIN  → 0007 → 0004 → 0002
SINUS  → 006  → 0003 → 0001

```

On notera que toutes les expressions binaires sont transcrites sur les listes de sortie en code hexadécimal.

PROGRAMME -ALPHA- ASSEMBLE PAR CAB 502B OCT 63 3

EMPL.	CODE OBJET	ADR1	NO.PS		PROGRAMME	SOURCE
0000	00000000		1	ALPHA	DFC	F=()
0001	9A 9A800004	0000	2	DEBUT	C	SINUS,4
0002	9A 9AA00004	0000	3		C	COSIN,5
0003	18 18A00014	0001	4		T	SINUS,5
0004	18 18800024	0002	5		T	COSIN,4
0005	C8 C8004B4A	0009	6		BCI	STOP,3,4
0006	9A 9AC00034	0003	7	ΔTG	C	SINUS,6
0007	0C 0CC00044	0004	8		DVF	COSIN,6
0008	1A 1AC00004	0000	9		TRZ	ALPHA,6
0009	38 38000000		10	STOP	HSE	
0001			11		FIN	DEBUT

SYMBOLES

ALPHA	0000
DEBUT	0001
STOP	0009

DEFINITIONS EXTERNES

TG	0006
----	------

REFERENCES EXTERNES

COSIN	0007
SINUS	0006

DIAGNOSTIC	()
------------	----

VI - EXPRESSIONS SYMBOLIQUES

Un autre avantage du système est de permettre l'utilisation d'expressions symboliques en partie adresses des instructions désignant la mémoire centrale comme contenu du 1er opérande.

1 - Composition

L'expression symbolique est constitué des symboles suivants, reliés par les opérateurs + ou -

- symboles alphanumériques : ALPHA, XB3, F12
- constantes décimales ou hexadécimales : 3200... X = B3F1
- symbole d'emplacement actuel : Δ

il représente la valeur du compteur d'adresse au moment de l'assemblage de l'instruction, il est encore appelé "symbole d'auto-adressage relatif".

2 - Translatabilité

Un programme est dit "translatable" s'il peut être placé, en vue de son exécution, en un endroit quelconque de la mémoire, il est alors rédigé en adresses relatives, par opposition aux programmes non translatables utilisant des adresses absolues. Il suffit que le système ajoute, lors du chargement, l'adresse de base aux adresses effectivement translatables pour que le programme soit exécutable.

Les symboles translatables sont évidemment les symboles alphanumériques et le symbole Δ , toutes les constantes représentant par définition des adresses absolues ne sont pas translatables.

Il est facilement vérifiable qu'une expression arithmétique est translatable si le nombre exprimant la différence entre les symboles translatables additionnés et les symboles translatables soustraits est égal à 1. Elle n'est pas translatable si cette différence est nulle, elle est illégale dans tous les autres cas.

Exemples

BETA + 64	TRANSLATABLE
BETA + ALPHA	ILLEGALE
BETA + ALPHA - SIGMA	TRANSLATABLE
Δ + X = F6	TRANSLATABLE
BETA - GAMMA	NON TRANSLATABLE
2800	NON TRANSLATABLE

3 - Légalité de l'expression arithmétique

Supposons qu'une expression arithmétique puisse contenir un ou plusieurs symboles externes ou "non encore définis", pour rester dans la généralité il faut admettre que ce ou ces symboles peuvent être utilisés à d'autres endroits du programme et qu'ils nécessitent un chaînage.

La méthode de liste enchaînée des références externes impose de conserver les adresses des derniers emplacements d'utilisation de chaque référence. L'instruction en cours d'assemblage ne peut contenir qu'une adresse ; il faut par ailleurs conserver la valeur de l'expression calculée avec les symboles éventuellement connus et la façon de calculer l'adresse finale à partir des symboles qui vont être définis ultérieurement.

Ces difficultés interviennent parce que l'assemblage est fait en un passage. Il n'est donc pas possible d'appliquer sans complication importante la méthode définie auparavant. L'avantage présenté par la programmation de telles instructions est faible en regard des complications introduites dans l'assemblage, d'autant plus que la probabilité d'utilisation est réduite.

En conséquence, toute expression symbolique ne peut contenir de terme externe ou "non encore défini" lors de sa génération par l'assembleur. Si l'instruction utilise un tel symbole, il doit être employé seul.

Après la définition du processus général d'assemblage et du langage symbolique, l'étude détaillée du travail et de l'organisation de l'assembleur fait l'objet du chapitre suivant.

NOTA : La machine à écrire comporte deux positions possibles pour la frappe des caractères ; le jeu des positions "haut" et "bas" permet de doubler le nombre de caractères utilisables pour un nombre donné de touches. Toutefois, l'utilisation des codes "haut" et "bas" peut mener à des erreurs et le langage symbolique est défini de manière à ne faire appel qu'à des caractères obtenus en position normale de la machine à écrire. L'utilisation d'un code "bas" dans la rédaction d'une instruction, serait considérée comme illégale par l'assembleur. Les commentaires, qui ne sont jamais analysés, peuvent évidemment utiliser tous les caractères possibles.

CHAPITRE 3

ETUDE ET CONCEPTION DE L'ASSEMBLEUR

L'assemblage n'est, dans la plupart des cas, qu'une étape dans le travail qui mène à l'exécution du programme binaire absolu obtenu par le chargement (en un endroit déterminé de la mémoire) du programme binaire translatable généré par l'assembleur.

Ainsi certaines phases de l'assemblage concernent plus directement le chargeur, et le choix des méthodes de traitement constitue déjà une définition de l'algorithme de chargement.

I - REMARQUES PRELIMINAIRES

La description des diverses parties du système sera effectuée de la façon suivante :

- explication du fonctionnement, choix des méthodes
- organigrammes divers
- programmes rédigés en langage autocode CAB 500

1 - Organigrammes

Deux types d'organigrammes seront généralement utilisés :

- l'organigramme fonctionnel traduit le procédé de traitement tandis que
- l'organigramme machine est l'étape nécessaire avant la rédaction du programme.

2 - Programmes

Afin de faciliter la compréhension, les programmes sont rédigés avec des repères pour les instructions et des commentaires.

Il est bien entendu que ces renseignements ne font pas partie des programmes fournis à la machine qui n'admet que les instructions "autocodes" situées dans la partie médiane des programmes présentés. Les adresses non affectées, sont les adresses de références externes à la section rédigée ; elles ne sont connues que lors de la mise en place des divers segments en mémoire et ne peuvent être désignées auparavant puisque l'emploi de symboles est interdit. Elles sont indiquées dans la zone commentaire par leur adresse relative dans la section de programme qui les définit et sont soulignées pour rappeler qu'elles n'ont pu être affectées qu'ultérieurement.

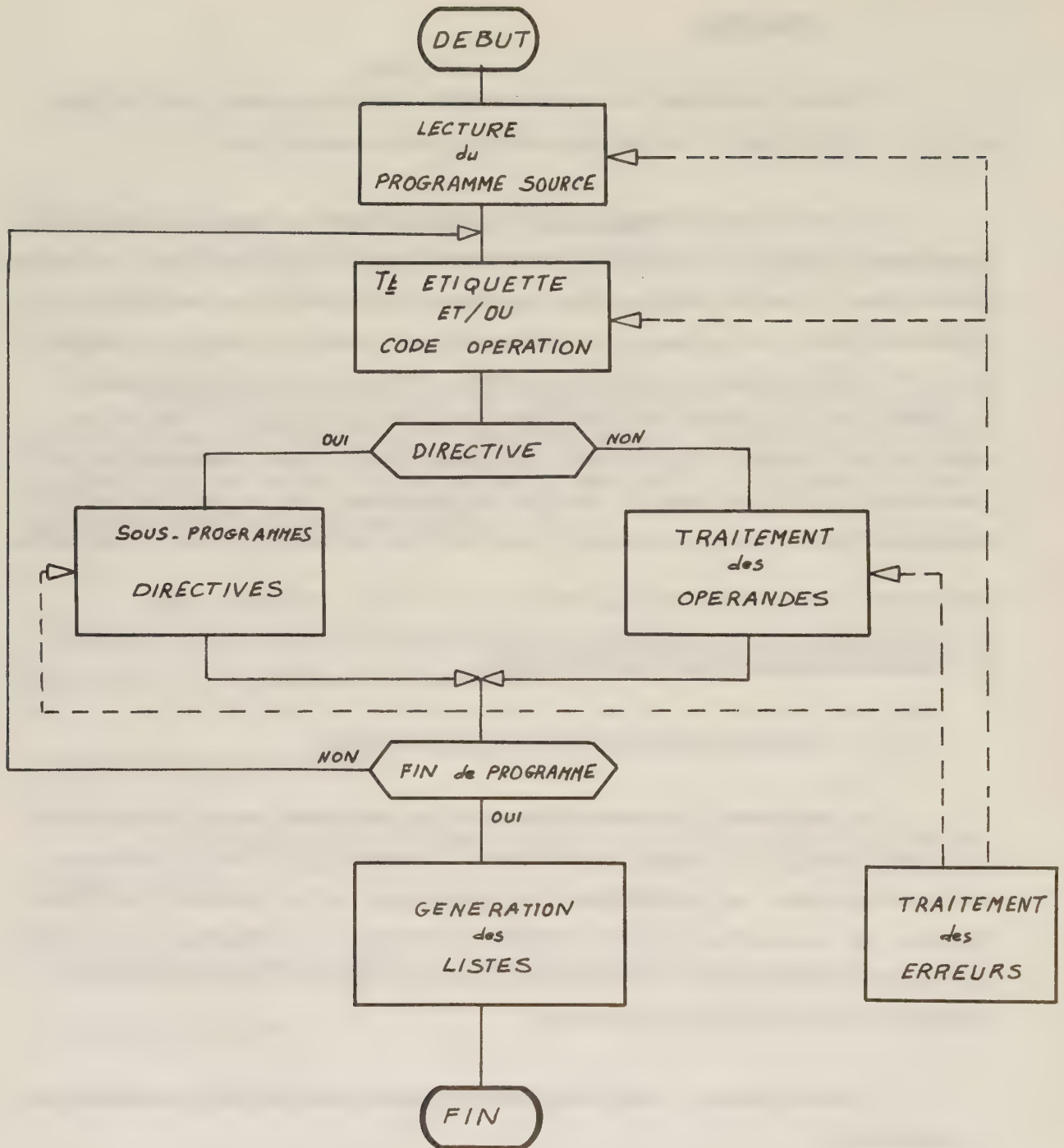
II - ALGORITHME GENERAL DE L'ASSEMBLAGE

1 - Travail et possibilités de l'assembleur

Le programme d'assemblage part d'un programme source rédigé dans le langage défini au chapitre précédent, il effectue la translittération des instructions, les générations de constantes et les réservations éventuelles. Par ailleurs, il doit préparer le travail du chargeur, en générant les divers renseignements nécessaires (table programme objet, table des références et définitions externes).

Dès maintenant, nous définissons trois possibilités au niveau de l'assemblage :

- a) - assemblage du programme source et édition de la liste imprimée, comprenant les programmes source et objet et les tables des symboles
- b) - assemblage, édition de la liste imprimée, génération de la bande objet binaire
- c) - assemblage et génération de la bande binaire seule.



1_ ASSEMBLAGE - Organigramme Simplifié

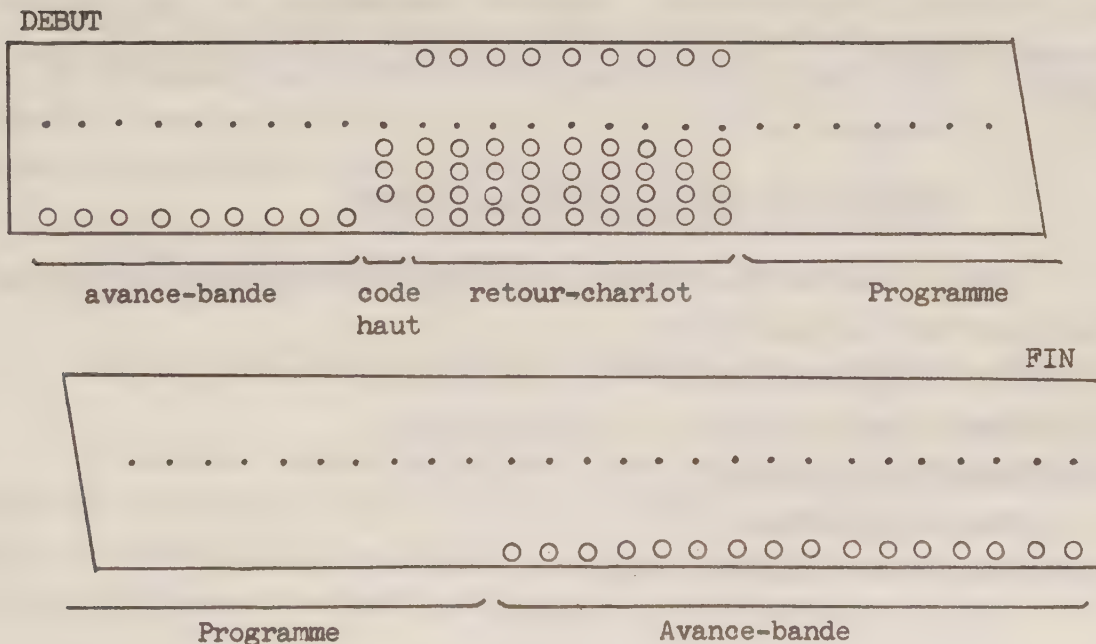
Ces diverses possibilités sont choisies par le programmeur lors de l'appel au système ; le fonctionnement est orienté par la partie "moniteur" du chargeur.

2 - Ruban source

Le programme initial peut éventuellement être formé directement à partir du clavier de la machine à écrire, mais les erreurs de frappe ne peuvent être corrigées et conduisent alors à des instructions erronées.

Il est donc préférable de perforer le programme sur bandes pour lesquelles les corrections sont relativement aisées. Une amélioration possible serait, lors de l'introduction du programme au clavier, d'envoyer la ligne dans une zone de mémoire effaçable validée par le retour chariot (en cas d'erreur, possibilité d'effacement du tout ou retour arrière pour correction).

Le "ruban source" sera conventionnellement établi de la façon suivante :



- Le début de bande est détecté par la présence d'un ou plusieurs codes "haut" qui assurent de trouver la machine à écrire en bonne position pour la frappe du programme ; les codes "RC" (retour de chariot) en nombre quelconque permettent au programmeur d'identifier facilement le début de bande
- La fin de programme (ou de bande) est constituée par la succession d'au moins dix codes "AVANCE-BANDE" (moins de dix codes sont ineffectifs).

3 - Organigramme

Le programme stocké en mémoire est translitéré instruction par instruction, après traitement du code opération le système se débranche aux sous-programmes "DIRECTIVES" ou au contraire au traitement des opérandes des instructions normales. Quand la fin du programme est reconnue, les différentes listes sont éventuellement éditées.

En cours d'assemblage, si le système détecte des erreurs, il fait appel à la séquence "traitement des erreurs" qui élabore les diagnostics et qui, suivant le type de faute, interdit l'assemblage de l'instruction (opérandes illégaux) ou arrête l'assemblage (débordement table des symboles).

4 - Segmentation des programmes du système

Pour l'étude et la rédaction des programmes constituant l'assembleur, les sections sont les suivantes :

P0 : lecture du programme source

P1 : traitement des étiquettes et codes opération

P2 : traitement des opérandes

P3 : traitement des erreurs

P4 : génération des listes de sortie

et les sous-programmes correspondant aux diverses directives (pseudo-instructions).

P0, P1, P2 et P3 constituent la 1ère phase de l'assemblage, P4 la seconde.

III - LECTURE DU PROGRAMME SOURCE

1 - A priori, deux solutions sont possibles :

- a) - lecture du programme et stockage en mémoire en vue des opérations ultérieures.
- b) - lecture et décodage simultanés des instructions, chargement en mémoire au fur et à mesure.

La deuxième solution présente l'avantage de travailler sur les caractères dès leur lecture et de diminuer le nombre d'échanges entre registres généraux et mémoire centrale, mais elle a l'inconvénient d'occuper les organes d'entrée pendant toute la durée de l'assemblage.

Nous choisissons donc la première méthode puisque de toute façon le programme source doit être disponible pour l'édition de la liste de sortie et que nous voulons éviter une relecture du programme qui demanderait des manipulations de bande perforée.

2 - Mise en place de la table "programme source"

Par la suite, cette table sera désignée par le symbole : TABPR.

La première adresse de la table étant connue de l'assembleur, le chargement est effectué par mot de cinq caractères. Quand le mot est complet, il est transféré en mémoire ; la fin d'instruction est testée par la présence du caractère "RC". Le dernier mot de l'instruction est

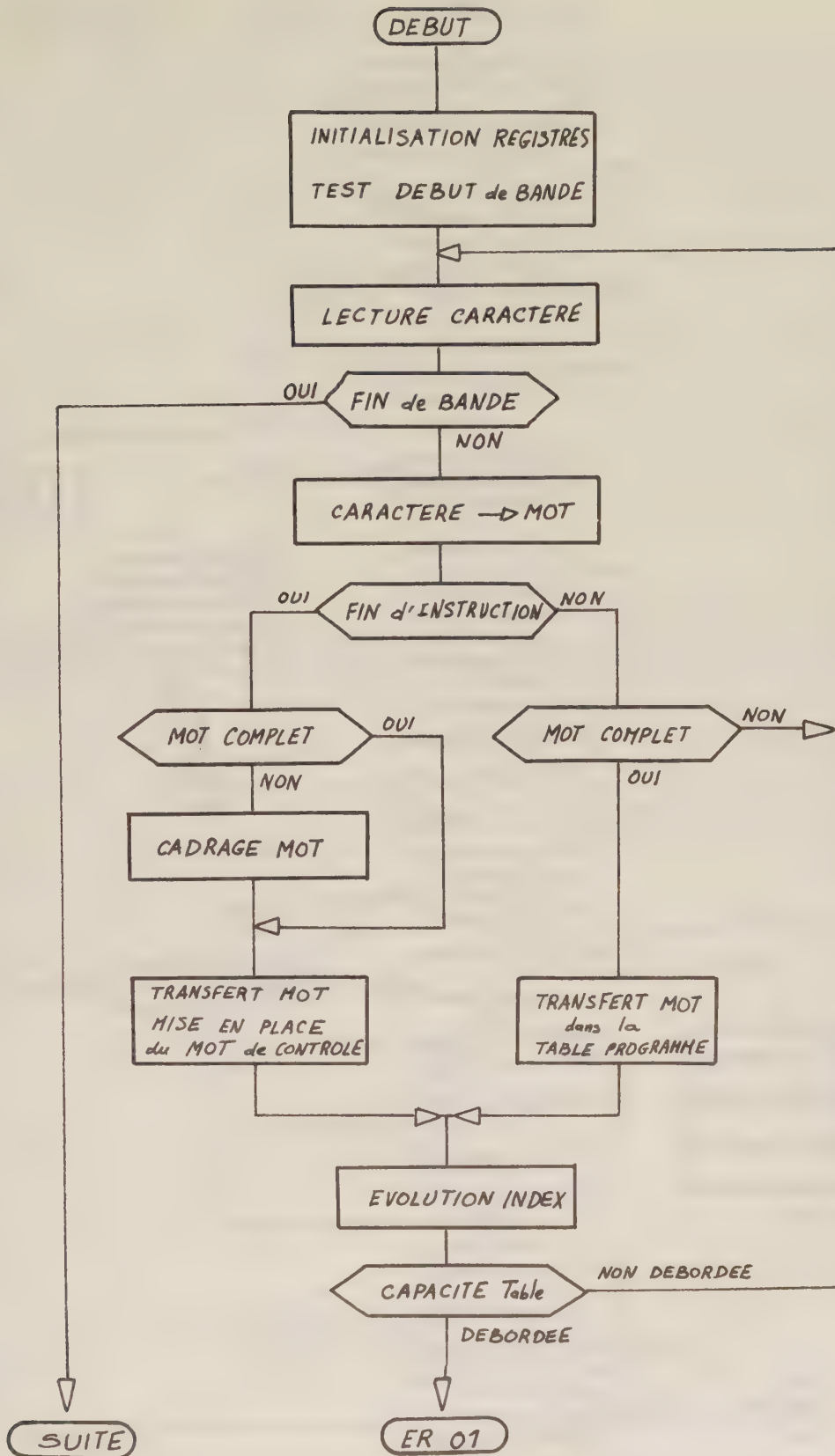
éventuellement complété par des codes "EFFACEMENT" et la mémoire suivante est réservée, elle a une double utilité :

- a) - lorsque l'assembleur ignore un commentaire, il n'a pas besoin d'explorer toute la chaîne des caractères jusqu'au "RC" indiquant la fin de ligne de programme, il lui suffit de tester les digits b31 et b32 des mots successifs.
- b) - elle peut contenir des renseignements complémentaires obtenus au cours de l'assemblage.

Les mémoires de la table contenant cinq caractères ont toujours leurs digits b31 et b32 nuls ; en plaçant systématiquement dans les mots de séparation le code binaire 11 en b31 et b32, le test d'identification se réduit à un test de signe.

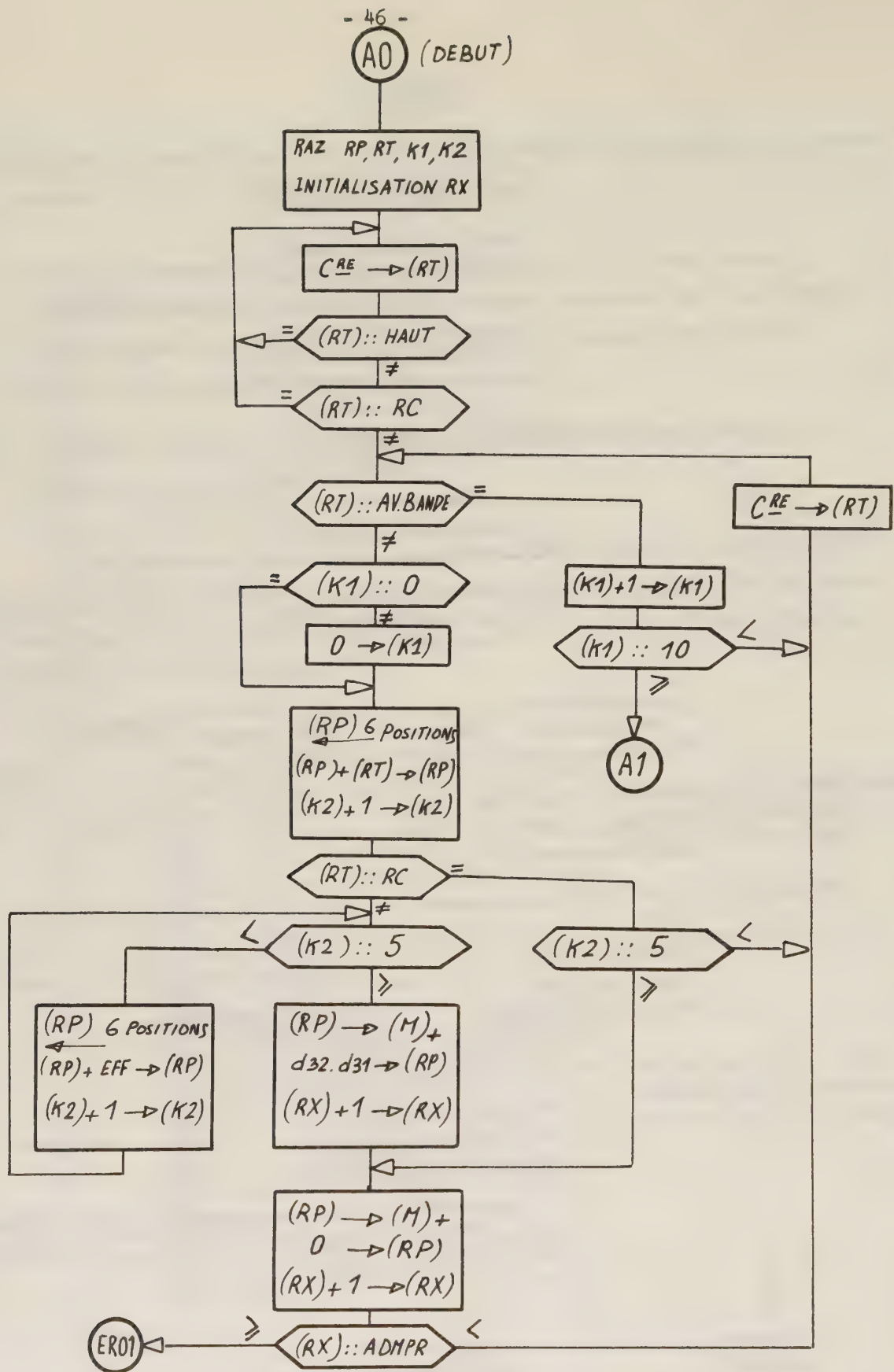
TABPR		C ₁	C ₂	C ₃	C ₄	C ₅	
		C ₆	C ₇	-	-	-	
→	00	-	-	-	-	-	Mots-instruction: 5 caractères DCB [b32=b31 = 0] contenu du mot ≥ 0
	00	-	-	-	-	-	
	00	-	-	-	-	-	
	00	-	-	-	-	-	
	00	-	-	-	-	"RC"	
→	11						Mot de séparation : b32 = b31 = 1 contenu du mot < 0
	00	-	-	-	-	-	
	00	-	-	-	-	-	
	00	-	-	-	-	-	
	00	-	-	-	-	-	
→	00	-	"RC"	"EFFt"	"EFFt"	"EFFt"	Mots-instruction complétés à 5 caractères par des codes "effacement"
	11						
	00	-	-	-	-	-	
	00	-	-	-	-	-	
	00	-	-	-	-	-	
instructions en DCB →	00	-	-	-	"RC"	"EFFt"	
	11						
	00	-	-	-	-	-	
	00	-	-	-	-	"RC"	
	11						
	00						

_ Table programme source _



LECTURE du PROGRAMME SOURCE -

2 - ASSEMBLEUR PO - Organigramme Fonctionnel



LECTURE du PROGRAMME SOURCE -

3- ASSEMBLEUR PO - ORG. Machine

ASSEMBLEUR PO : LECTURE DU PROGRAMME SOURCE

0	A0EV0	RAZ RP,K1,K2	
	A1EV0		
	A2EV0		
	A15EM5(Δ)	A (1ère ADRESSE TABPR)	(3ΔP1)
4	A3EL0S	LECTURE	
	X		
	V3TV24	TEST "HAUT"	
	RV4ΔZ13		
	V3TV25	TEST "RC"	
	RV4ΔZ13		
10	V3TV32	TEST "AVANCE-BANDE"	
	RV16ΔY13		
	S1V10	TEST K1	
	RV46ΔP1		
	A1V11		
	RV43Δ		
16	A1EV0Y1		
	K0QV6	CHARGEMENT CARACTERE	
	T14R0		
	A0R3		
	A2V1		
	V3QV25	TEST "RC"	
	RV26ΔZ14		
	V2TV5	TEST "K2 "	
	RV43ΔY13		
	RV36Δ		
26	V2TV5	TEST "K2 "	
	RV33ΔZ13		
	K0QV6	CADRAGE MOT	
	T14R0		
	A0V31	31 = EFFACEMENT	
	A2V1		
	RV26Δ		
33	T0M0+	TRANSFERT INDEXE	
	A0EW49152	B32=B31=1	
	A15V1		
	T0EM0+		
	A15V1		
	T15R13		
	S13M4(ΔS		
	X	ADRESSE MAX+1 (TABPR)	
	RV965ΔP13	A (ER 01)_DEBORDEMENT TABPR	(0Δ P3)
	A2EV0		
43	A3EL0S		
	X		
	RV10Δ		
46	T15EM832Δ	A (DERNIERE ADRESSE PROGRAMME)	(597 Δ P2)
	FE	FIN DE BANDE PROGRAMME	

3 - Organigrammes et programme

La lecture alphanumérique n'est programmable que caractère par caractère bien que la machine soit conçue pour utiliser des mots. La génération du tableau conduit donc au travail suivant :

- chargement du caractère dans un registre (RT) où il peut être testé
- transfert dans le registre (RP) où sont générés les "mots programme" successifs
- décalage du registre RP de six positions à gauche et addition du registre RT, et ainsi de suite.

Le nombre de caractères du mot est donné par le registre K2, le registre K1 étant utilisé pour la totalisation des caractères "avance-bande". Dans le cas où le programme source ne pourrait entrer dans la zone prévue en mémoire, la lecture est arrêtée et la commande est donnée à la section "ER01" qui est chargée du traitement d'un tel cas. (cf. chapitre 3 paragraphe VII . Traitement des erreurs).

Le choix des codes "effacement" pour compléter les mots est dicté par le fait qu'ils ne correspondent à aucune opération en impression et qu'ils sont ignorés à la lecture d'une bande perforée.

IV - CONFIGURATIONS ET GESTION DES TABLES

Assembleur et chargeur doivent posséder certains renseignements, pour systématiser les recherches ceux-ci sont groupés dans des tables ayant chacune une utilité particulière.

1 - Table des symboles

- a) - C'est en analysant étiquettes et symboles que l'assembleur construit cette table, elle doit contenir d'une part les symboles utilisés dans le programme et d'autre part les adresses correspondantes. Nous

avons vu que ces adresses ne sont pas forcément définies et que plusieurs types de symboles existent ; pour faciliter le traitement nous les groupons tous dans la même table, l'éclatement en divers sous-ensembles est fait à la fin de l'assemblage suivant les besoins du programme d'édition et du chargeur.

Le rangement des symboles par ordre alphabétique n'est fait qu'après éclatement de la table diminuant ainsi le nombre moyen de comparaisons à effectuer pour le classement. Par ailleurs, l'ordre alphabétique n'est pas nécessaire lors de la 1ère phase de l'assemblage et cela imposerait de reclasser la table à chaque nouveau symbole détecté.

- b) - La table des symboles est constituée de la manière la plus simple pour éviter les appels "mémoire" qui sont longs (tambour), sa configuration est dite en "tableau mixte" (figure 3).

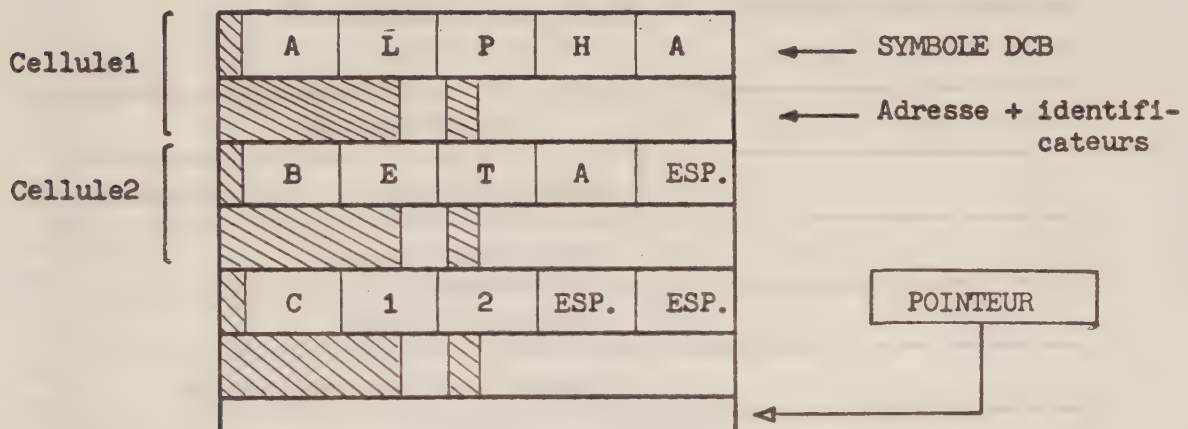
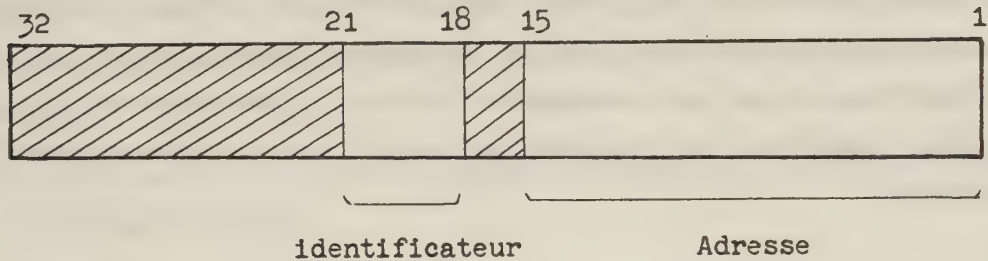


Figure 3

L'adresse de tête du tableau "TABSB" est connue du système et la première adresse libre est tenue à jour par un registre utilisé en pointeur.

La première mémoire d'une cellule contient le symbole (en caractères DCB) cadré à droite ; des codes "espace" remplissent les places vides.

La deuxième mémoire contient l'adresse correspondante et un code identificateur :



Le code identificateur est donné par le tableau 4

b21	b20	b19	b18	SIGNIFICATION
0	0	0	0	Symbole défini par une étiquette
0	0	1	0	Définition externe
0	1	0	0	Référence externe
1	0	0	0	Symbole multiplement défini
0	0	0	1	Symbole non défini

Figure 4 - Identificateur de symbole

Nous avons déjà étudié le problème des définitions et références externes ; un symbole est évidemment multiplement défini s'il existe dans le programme plusieurs lignes ayant la même étiquette.

ASSEMBLEUR PO : LECTURE DU PROGRAMME SOURCE

0	A0EV0	RAZ RP,K1,K2	
	A1EV0		
	A2EV0		
	A15EM50Δ	A (1ère ADRESSE TABPR)	(3ΔP1)
4	A3ELO8	LECTURE	
	X		
	V3TV24	TEST "HAUT"	
	RV4ΔZ13		
	V3TV25	TEST "RC"	
	RV4ΔZ13		
10	V3TV32	TEST "AVANCE-BANDE"	
	RV16ΔY13		
	S1V10	TEST K1	
	RV46ΔP1		
	A1V11		
	RV43Δ		
16	A1EVOY1		
	K0QV6	CHARGEMENT CARACTERE	
	T14R0		
	A0R3		
	A2V1		
	V3QV25	TEST "RC"	
	RV26ΔZ14		
	V2TV5	TEST "K2"	
	RV43ΔY13		
	RV36Δ		
26	V2TV5	TEST "K2"	
	RV33ΔZ13		
	K0QV6	CADRAGE MOT	
	T14R0		
	A0V31	31 = EFFACEMENT	
	A2V1		
	RV26Δ		
33	T0M0+	TRANSFERT INDEXE	
	A0EW49152	B32=B31=1	
	A15V1		
	T0EM0+		
	A15V1		
	T15R13		
	S13M40ΔS		
	X	ADRESSE MAX+1 (TABPR)	
	RV965ΔP13	A (ER 01)_DEBORDEMENT TABPR	(0Δ P3)
	A2EV0		
43	A3ELO8		
	X		
	RV10Δ		
46	T15EM832Δ	A (DERNIERE ADRESSE PROGRAMME)	(597 Δ P2)
	FE	FIN DE BANDE PROGRAMME	

3 - Organigrammes et programme

La lecture alphanumérique n'est programmable que caractère par caractère bien que la machine soit conçue pour utiliser des mots. La génération du tableau conduit donc au travail suivant :

- chargement du caractère dans un registre (RT) où il peut être testé
- transfert dans le registre (RP) où sont générés les "mots programme" successifs
- décalage du registre RP de six positions à gauche et addition du registre RT, et ainsi de suite.

Le nombre de caractères du mot est donné par le registre K2, le registre K1 étant utilisé pour la totalisation des caractères "avance-bande". Dans le cas où le programme source ne pourrait entrer dans la zone prévue en mémoire, la lecture est arrêtée et la commande est donnée à la section "ERO1" qui est chargée du traitement d'un tel cas. (cf. chapitre 3 paragraphe VII . Traitement des erreurs).

Le choix des codes "effacement" pour compléter les mots est dicté par le fait qu'ils ne correspondent à aucune opération en impression et qu'ils sont ignorés à la lecture d'une bande perforée.

IV - CONFIGURATIONS ET GESTION DES TABLES

Assembleur et chargeur doivent posséder certains renseignements, pour systématiser les recherches ceux-ci sont groupés dans des tables ayant chacune une utilité particulière.

1 - Table des symboles

- a) - C'est en analysant étiquettes et symboles que l'assembleur construit cette table, elle doit contenir d'une part les symboles utilisés dans le programme et d'autre part les adresses correspondantes. Nous

avons vu que ces adresses ne sont pas forcément définies et que plusieurs types de symboles existent ; pour faciliter le traitement nous les groupons tous dans la même table, l'éclatement en divers sous-ensembles est fait à la fin de l'assemblage suivant les besoins du programme d'édition et du chargeur.

Le rangement des symboles par ordre alphabétique n'est fait qu'après éclatement de la table diminuant ainsi le nombre moyen de comparaisons à effectuer pour le classement. Par ailleurs, l'ordre alphabétique n'est pas nécessaire lors de la 1ère phase de l'assemblage et cela imposerait de reclasser la table à chaque nouveau symbole détecté.

- b) - La table des symboles est constituée de la manière la plus simple pour éviter les appels "mémoire" qui sont longs (tambour), sa configuration est dite en "tableau mixte" (figure 3).

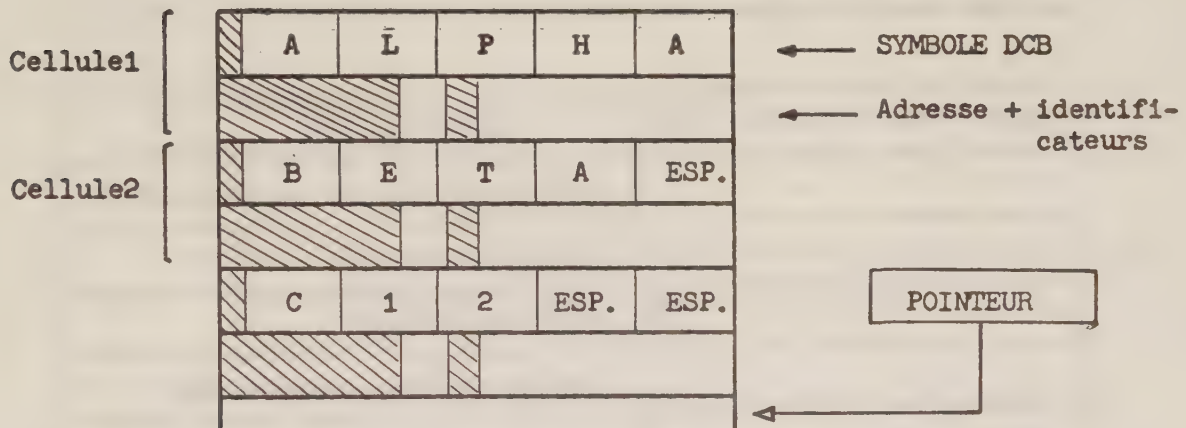
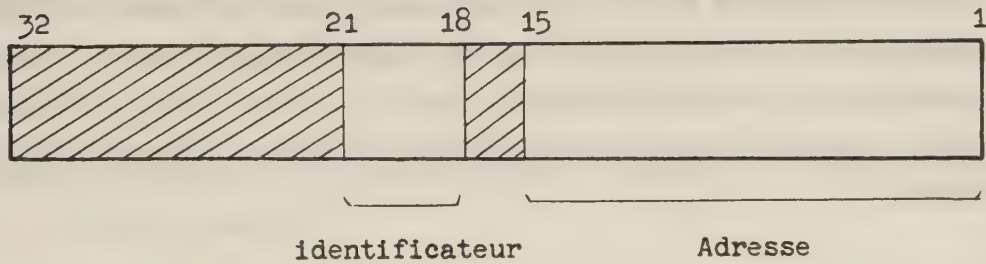


Figure 3

L'adresse de tête du tableau "TABSB" est connue du système et la première adresse libre est tenue à jour par un registre utilisé en pointeur.

La première mémoire d'une cellule contient le symbole (en caractères DCB) cadré à droite ; des codes "espace" remplissent les places vides.

La deuxième mémoire contient l'adresse correspondante et un code identificateur :



Le code identificateur est donné par le tableau 4

b21	b20	b19	b18	SIGNIFICATION
0	0	0	0	Symbole défini par une étiquette
0	0	1	0	Définition externe
0	1	0	0	Référence externe
1	0	0	0	Symbole multiplement défini
0	0	0	1	Symbole non défini

Figure 4 - Identificateur de symbole

Nous avons déjà étudié le problème des définitions et références externes ; un symbole est évidemment multiplement défini s'il existe dans le programme plusieurs lignes ayant la même étiquette.

Le cas du symbole non défini ne correspond qu'aux instructions de branchement ; à la fin de la 1ère phase tous les symboles non définis sont classés références externes.

- c) - Le nombre maximum de symboles est fixé à 128, mais cette quantité peut facilement être modifiée car elle constitue une donnée pour le programme d'assemblage.
- d) - La recherche dans une telle table sera séquentielle car l'ordre des symboles est normalement quelconque.

2 - Tables des codes opération (annexe n° 9)

- a) - Le nombre d'opérations possibles est de 70 environ, le traitement par chaîne de tests serait trop long et trop laborieux ; il est préférable d'établir une table des codes opérations permettant d'obtenir directement le code binaire correspondant.

Toutefois la recherche par balayage dans une telle table nécessite un temps moyen de 700 ms (35 appels tambour)!

La consultation de table par dichotomie se justifie pleinement quand le nombre de termes est une puissance entière de deux.

Le nombre étant quelconque, nous avons pensé à utiliser une méthode mixte faisant intervenir une table des matières (ceci revient à créer une partition dans l'ensemble des codes opération utilisés).

- b) - La table des codes opération (figure 5), regroupe les codes débutant par la même lettre et la recherche s'effectue par balayage dans les sous-ensembles ainsi définis. Le sous-tableau le plus important (lettre D) comporte dix opérations et le temps moyen de consultation est réduit à 100 ms. Pour abaisser encore le temps de traitement les

codes sont placés, dans chaque classe, en fonction de leur probabilité d'utilisation (déterminée par une étude statistique sur un nombre important d'instructions machine . cf. annexe n°6).

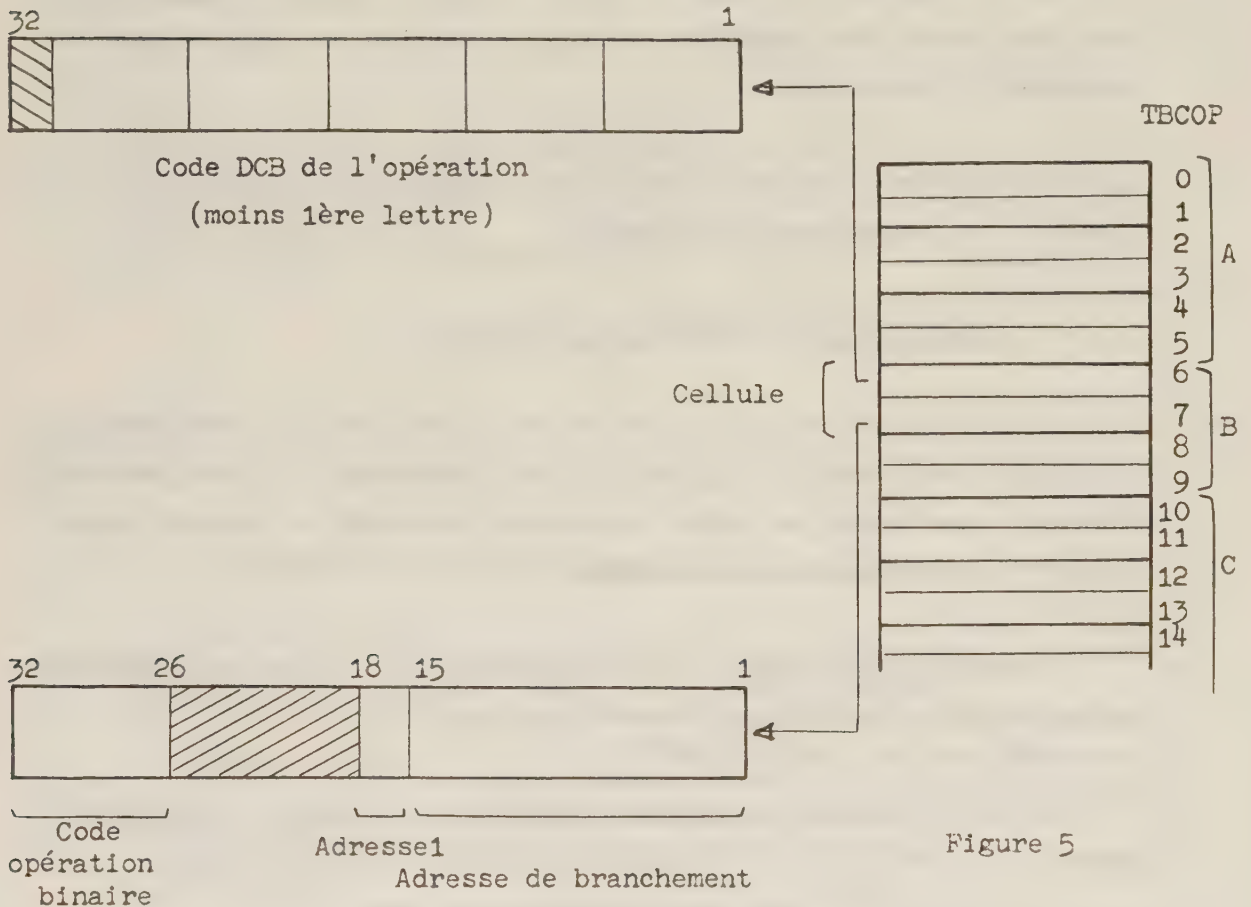


Figure 5

La table est constituée de cellules de deux positions. La première contient le code DCB de l'opération, sans sa première lettre qui constitue la fonction de partition ; la deuxième mémoire contient :

- digits 1 à 15 : adresse de branchement de la suite du traitement, elle dépend évidemment du type d'instruction. Si le contenu de cette zone est nul, il s'agit d'une instruction sans opérande.
- digits 16 à 17 : ce champ indique la possibilité d'adressage du 1er opérande

A certaines lettres ne correspond aucun code opération, conventionnellement le contenu de la mémoire est nul et cela indique à l'assembleur que le code utilisé n'existe pas.

Exemple : soit à rechercher un code opération dont la première lettre est B

$$\text{Adr} = \text{TBMAT} + \text{Code B} - \text{code A} = \text{TBMAT} + 1$$

La mémoire d'adresse (TBMAT + 1), donne la 1ère adresse du tableau B (TBCOP + 6) et la première adresse ne faisant plus partie de ce tableau (TBCOP + 10). La structure de la table des codes opération est celle correspondant à la figure 5 (paragraphe IV 2 b).

Remarque :

Le système d'assemblage est conçu de façon à être placé à n'importe quel endroit de la mémoire. Les adresses contenues dans les tables des codes opération sont relatives à l'adresse d'origine de l'assembleur, il peut ainsi assurer lui-même la translation de ces adresses (cf. "traitement des codes opérations").

3 - Table des instructions binaires (figure 7)

La traduction des différentes instructions et l'exécution des directives conduit à l'élaboration du programme objet binaire.

- a) - Ce programme est empilé dans un tableau dont l'adresse de tête est connue du système, l'adresse de la première mémoire libre est donnée par un pointeur. Le contenu de ce pointeur en fin de programme (pseudo-instruction "FIN") permet de déterminer la taille du programme généré: elle est placée dans la première position du tableau, la mémoire suivante contenant éventuellement le symbole DCB du nom du programme.

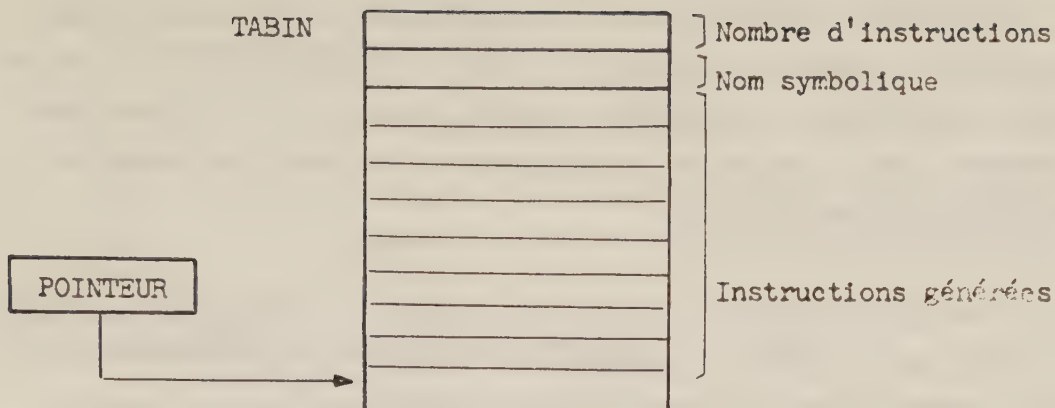


Figure 7

La table binaire comprend les instructions machine et les diverses constantes générées par les directives. Nous verrons dans l'étude particulière de chacune, la façon dont ces constantes sont placées dans le programme objet.

4 - Table des codes instruction

a) - Nécessité d'une telle table

Pour l'impression de la liste de sortie et surtout pour le chargement, la nature du contenu de chaque mot binaire de la table doit être précisée (à savoir : instruction machine ou ordre de gestion chargeur ; adresse absolue, translatable ou chaînage).

L'étude du langage machine CAB 500 a montré que la totalité du mot est utilisée dans la majorité des instructions, il n'est donc pas possible de placer ces indications dans les constructions binaires elles-mêmes. Il faut donc affecter à chaque mot binaire, un certain nombre de digits de contrôle qui en spécifient la nature : c'est le rôle de la table des codes instruction.

La CAB 500 ne possède pas d'instruction de manipulation de caractères. La première solution qui vient à l'esprit est de placer un mot de contrôle par instruction binaire, cela double la place prise en mémoire par le programme objet alors que la totalité du mot de contrôle n'est pas utilisée.

Il est plus rationnel de prendre un champ de digit dont la taille est fixée par le nombre d'éventualités à coder. Le caractère de 4 digits permet d'avoir 16 possibilités différentes (ce qui semble suffisant, au premier examen) et présente deux avantages :

- il est directement représentable par un chiffre hexadécimal.
- le nombre de caractères logeable dans un mot (8) est une puissance entière de 2 ; facilitant ainsi la détermination de la place d'un caractère dans la table (figure 8).

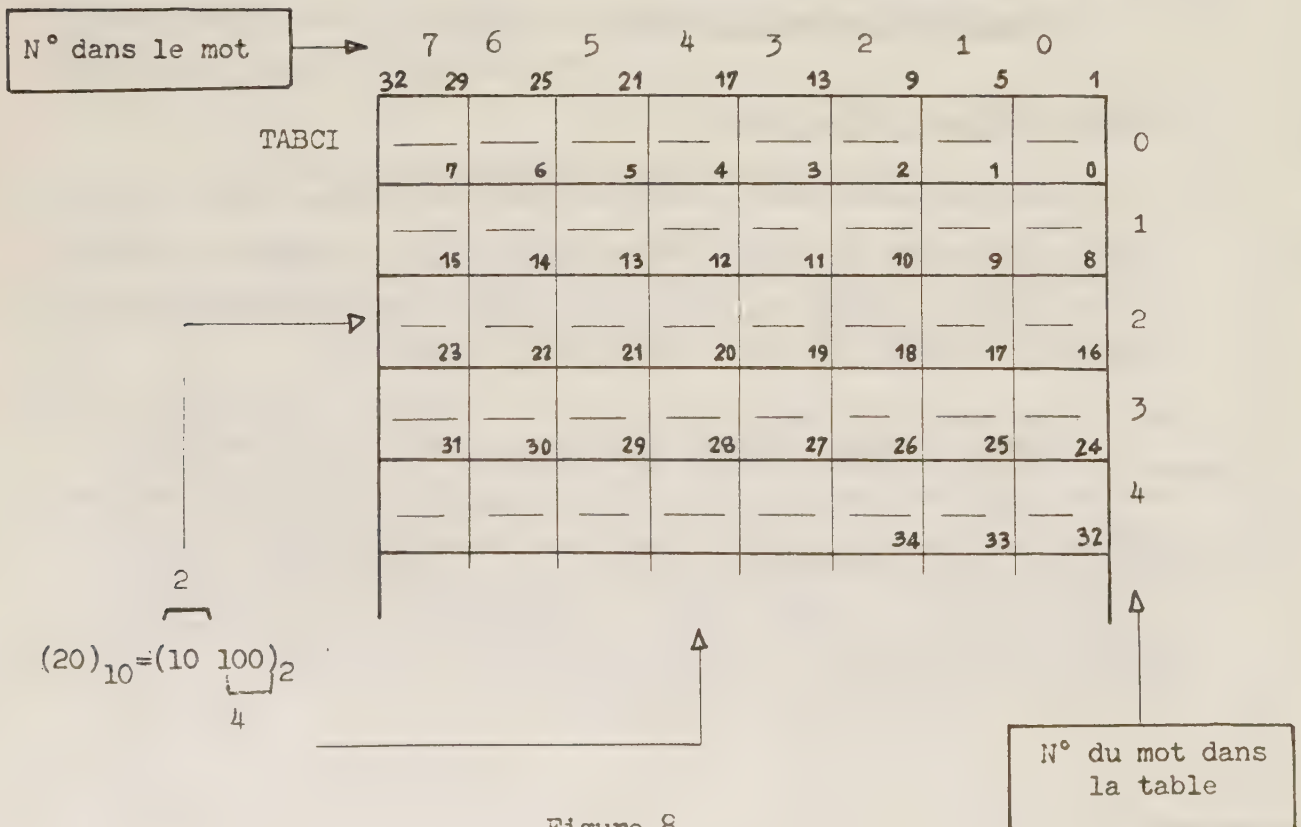


Figure 8

Exemple : soit à déterminer l'adresse du caractère correspondant à l'instruction binaire n° 20 :

$$(20)_{10} = (10100)_2$$

les trois derniers digits fournissent le numéro du caractère dans le mot, les autres digits fixant le numéro du mot dans la table.

b) - Manipulation des codes instruction

- "chargement d'un caractère":

le caractère à charger étant initialement cadré en b_1 , il suffit de le décaler à gauche d'un nombre de positions V donné par la relation :

$$\underline{V = 4 N}$$

$N = n^\circ$ du caractère dans le mot

et de l'ajouter au mot dont l'adresse est :

$$\underline{Adr = TABCI + M}$$

$M = n^\circ$ du mot dans la table

- "Modification d'un caractère" :

l'opération a lieu en deux temps :

1 - extraction du caractère et mise à zéro de la zone correspondante du mot

2 - chargement du nouveau caractère.

- "Extraction d'un caractère pour test" :

pour généraliser le test, on cadre toujours le caractère à tester en b_1 .

1 - extraction du caractère

2 - cadrage en b_1 par décalage à droite de la quantité $V = 4N$.

Toutes ces opérations de décalage peuvent être systématisées par l'emploi d'une table de décalage.

5 - Table de décalage (figure 9)

Les opérations de décalage dont le nombre de position est variable au cours du programme ne peuvent être réalisées par un opérande immédiat.

Le langage CAB 500 permet d'effectuer des décalages dont le nombre de positions est fixé par le contenu d'un registre ou d'une mémoire. C'est en quelque sorte un adressage indirect mais l'opérande doit être un digit placé dans une position particulière qui dépend du sens de décalage. (figure 9).

N	V	DECALAGE A GAUCHE $V' = V + 1$	DECALAGE A DROITE $V' = 33 - V$
0	0	b1	b33
1	4	b5	b29
2	8	b9	b25
3	12	b13	b21
4	16	b17	b17
5	20	b21	b13
6	24	b25	b9
7	28	b29	b5

Figure 9

Connaissant le numéro N du caractère à manipuler dans le mot, il suffit de faire un décalage indexé par la table représentée par la figure 9bis.

Le contenu du registre d'index est :

- pour un décalage à droite $(R\ x) = 8 - N$
- pour un décalage à gauche $(R\ x) = N$

		29	25	24	17	13	9	3	1
TABDC	0								1
	1							1	
	2						1		
	3					1			
	4				1				
	5			1					
	6		1						
	7	1							
	8								

- Figure 9 bis -

TABLE DE DECALAGE

Nota : l'utilisation de cette table est un détail de programmation qui a son importance, car elle intervient à plusieurs reprises dans l'assemblage.

6 - Table des erreurs (figure 10)

Cette table permet de conserver les diagnostics d'erreurs éventuellement émis en cours d'assemblage. Dès maintenant, nous pouvons préciser que les erreurs détectées dans une ligne de programme sont de deux types :

"ERREUR 1" : erreur concernant l'étiquetage de l'instruction ; elle n'intervient pas dans l'assemblage de cet ordre.

"ERREUR 2" : erreur dans la syntaxe de l'instruction elle-même (zone opération ou opérande) interdisant la traduction.

Afin d'éviter une consultation séquentielle de la table lors de l'écriture du listing, celle-ci ne contient que les codes indiquant la nature des erreurs détectées. Les codes des instructions erronées sont caractérisés par un drapeau d'erreur, et un compteur permet de faire appel à la cellule correspondante du tableau sans qu'il soit besoin d'y placer le numéro de l'instruction.

L'adresse de tête du tableau est fournie à l'assembleur et le nombre maximal de lignes de programme incorrectes est conventionnellement fixé à 128.

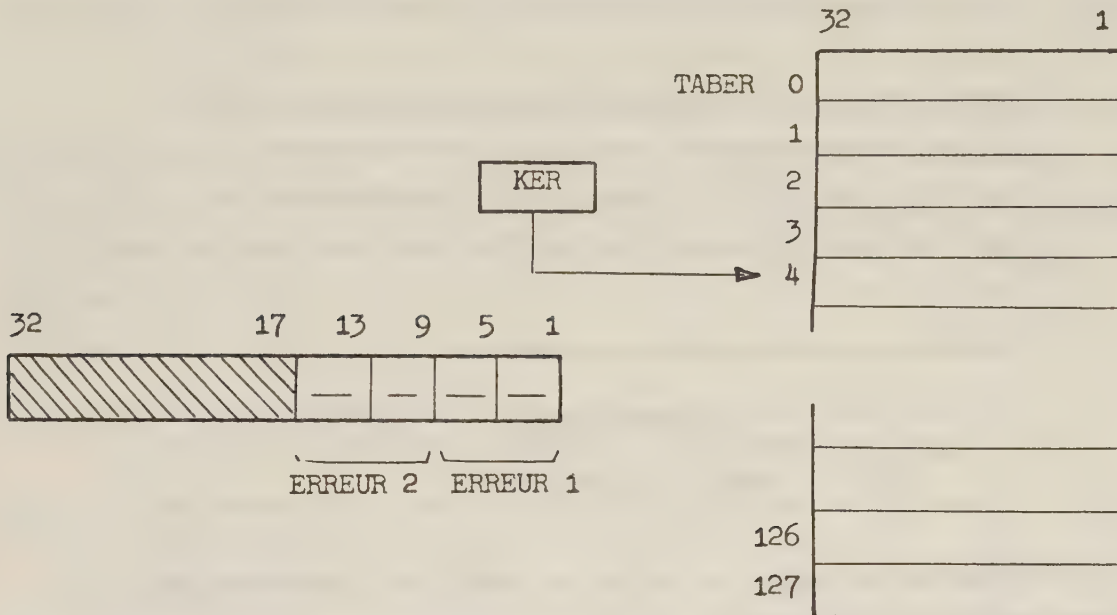


Figure 10 - TABLE DES ERREURS

Les digits 1 à 4 pour la classe 1 et 9 à 12 pour la classe 2 indiquent le code conventionnel de l'erreur détectée, tandis que les bits 5 à 8 et 13 à 16 rappellent le type d'erreur lors de l'écriture du diagnostic dans le listing. (La valeur zéro dans l'un des derniers champs considérés signifie l'absence d'erreur du type correspondant).

Exemple :

	0010	0111	0001	0010	erreur 12 et erreur 27
			0001	0011	erreur 13
	0010	0100			erreur 24

L'annexe n° 7 donne la signification des diagnostics pouvant être émis dans le listing d'assemblage. L'apparition du diagnostic : ERREUR 26 signifierait que l'instruction ne peut être assemblée parce qu'elle utilise une expression symbolique illégale du point de vue translatabilité.

V - TRAITEMENT DES ETIQUETTES ET DES CODES OPERATION

Cette section de programme comporte deux parties principales qui sont groupées parce qu'elles nécessitent toute deux la lecture d'une zone symbolique (étiquette - code opération).

1 - Traitement des étiquettes

En début d'assemblage le compteur d'adresses KA est remis à zéro et le pointeur de la table des symboles (PTRS) est chargé de la 1ère adresse de la table.

La lecture du 1er caractère de l'instruction à analyser constitue l'entrée réelle dans la séquence de traitement, c'est à cet endroit que le programme d'assemblage se rebranche à chaque nouvelle instruction (B1).

Le registre RP contient donc le caractère à tester, s'il s'agit d'une barre de fraction (commentaire) ou d'un retour-chariot (saut de ligne), le registre RTR (le registre "translatabilité" est utilisé pour générer le code instruction) est positionné à 16 et le programme se débranche à la séquence de P2 chargée d'ignorer le commentaire (C07). La présence de l'étiquette est indiquée par le fait que le 1er caractère de la ligne n'est pas une tabulation (caractère de séparation de zone) sinon il y a branchement à la partie "code opération" (C1). La lecture du symbole Δ (définition externe) met à 1 le registre R Δ et impose la lecture d'un nouveau caractère. Le début d'étiquette étant effectivement une lettre, le sous-programme de lecture symbole est appelé et, si le contrôle

du caractère de séparation est satisfait, le symbole est cadré à gauche dans le registre ROP pour faciliter le classement alphanumérique ultérieur.

Le symbole correspondant à l'étiquette est recherché dans la table des symboles :

- a) - le symbole n'est pas dans la table, la valeur du compteur d'adresse et le code DCB du symbole sont placés dans la table (D1).
- b) - le symbole est déjà dans la table (E1), la zone d'identification de la mémoire contenant l'adresse du symbole permet d'orienter le fonctionnement dans deux directions :
 - le symbole est déjà défini (b 18 = 0) et un diagnostic d'erreur est élaboré (F1)
 - le symbole n'est pas défini (b 18 = 1), la zone d'adresse indique le 1er emplacement où il faut charger la valeur du compteur d'adresse et le numéro du code instruction à modifier (le code qui désignait une instruction contenant une adresse de chaînage doit maintenant correspondre à une adresse translatable). La valeur du compteur d'adresse est évidemment placée dans la table des symboles. La mise à jour des instructions et des codes se poursuit éventuellement jusqu'à la rencontre de l'adresse nulle qui indique la fin de la chaîne.

A titre d'exemple, voyons l'évolution de la table des symboles, de la table binaire et des codes instruction au cours du programme défini par :

KA = 0003	instruction de branchement à l'adresse BETA
KA = 0005	instruction de branchement à l'adresse BETA
KA = 0006	instruction dont l'étiquette est △ ALPHA
KA = 0008	instruction dont l'étiquette est BETA

(KA) = 003

0				4	-	-	-
TABCI							

TABIN	
	0

0	B	E	T	A	-
TABSE1		1			3

$$\underline{(K_A) = 0.05}$$

TABCI	0		4	-	4	-	-	-
	1							

TABIN			
		0	-
		3	-

TAB33	B	E	T	A	-
		1			5

$$\underline{(KA) = 006}$$

TABCI	-	4	-	4	-	-	-
-------	---	---	---	---	---	---	---

TABIN			
		0	-
		3	-

TABSE	B	E	T	A	—
		1			5
	A	L	P	H	A
		2			6

(KA) = 0008

TABCI	-	-	0	-	0	-	-	-
								-

TABIN			
		8	-
		8	-

TABSE	B	E	T	A	—
		0			8
	A	L	P	H	A
		2			6

Les tirets représentent un contenu quelconque, les codes instructions sont supposés être 4 pour une adresse de chaînage et 0 pour une adresse translatable.

2 - Traitement des codes opération

Le caractère de séparation a été testé dans la première phase du traitement, le caractère suivant est lu dans le registre programme (RP) et doit être une lettre comprise entre A et T (aucun code opération ne commence par U, V, W, X, Y, Z) ; cette lettre est mise en garde dans le registre (RS) et le sous-programme de lecture place la suite du code dans le registre ROP. La dernière lettre du code qui définit le type de premier opérande est extraite et positionne le registre clé R Δ aux valeurs :

0	$\rightarrow$	(R Δ)	R	registre
B 16	$\rightarrow$	(R Δ)	quelconque	mémoire
B 17	$\rightarrow$	(R Δ)	I	immédiat

Le contenu de RS permet le calcul de l'adresse de la cellule de la table des matières qui fournit les adresses de tête et de queue du sous-tableau des codes opération correspondants.

Si le code opération existe dans la table, la mémoire suivant le symbole DCB est chargée dans ROP et scindée en trois parties.

- le champ b 16. b 17 par comparaison à R Δ décide de la compatibilité du code opération et du premier opérande.
- le champ b 26..b 32 définissant le code binaire de l'instruction est transféré dans le registre instruction (c'est le registre de formation de l'instruction à assembler).

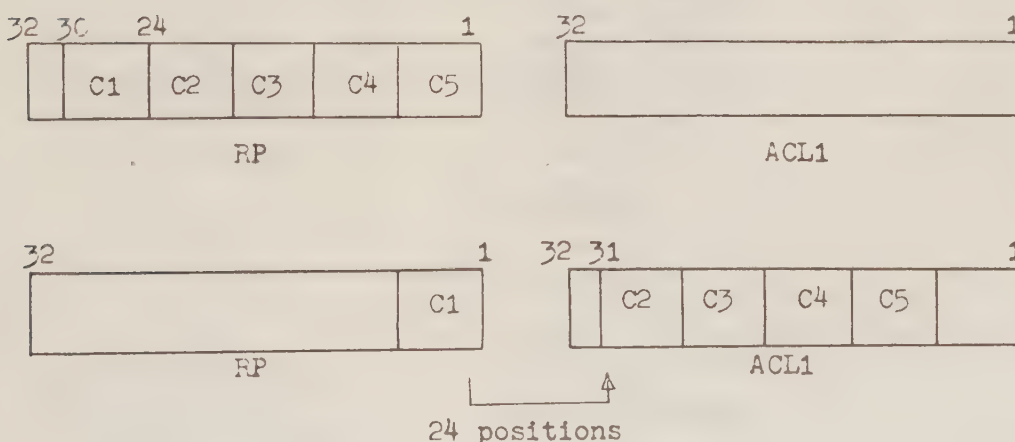
- les digits 1 à 15 fournissent l'adresse de la suite du traitement. Si cette zone est nulle, on passe directement à la fin d'instruction (M2) car il s'agit d'un ordre sans opérande.

3 - Programmation

- a) - L'analyse des champs d'instruction est faite par observation des caractères successifs et recherche des caractères de séparation. Cette méthode préférée à l'analyse à partir de positions définies pour chaque champ a conduit au choix du format de la ligne de programme utilisant le langage symbolique défini précédemment. Cette convention présente deux avantages:
- la place prise, par le programme source en mémoire est minimale
 - le caractère de séparation "tabulation" apporte une présentation très claire du programme écrit.

b) - Lecture d'un caractère

Nous avons déjà vu que le programme source est placé en mémoire à raison de 5 caractères par mot. L'analyse des caractères successifs impose l'éclatement du mot. Le mot est transféré dans le registre programme, et les caractères sont amenés en position de test (b 1 à b 6) par décalage à droite dans le registre double (RP, ACL1)



Lors de l'extraction du 1er caractère, le cadrage dans RP est en B 30, le décalage fait passer le digit 1 de RP en position 31 de ACL1, quand on replace les caractères restant dans RP le nouveau cadrage est en B31 et pour faire apparaître C2 en position de test le décalage doit être de 25 positions au lieu de 24. Le registre RL tient le compte des caractères lus dans le mot.

L'organigramme est celui de la figure 11

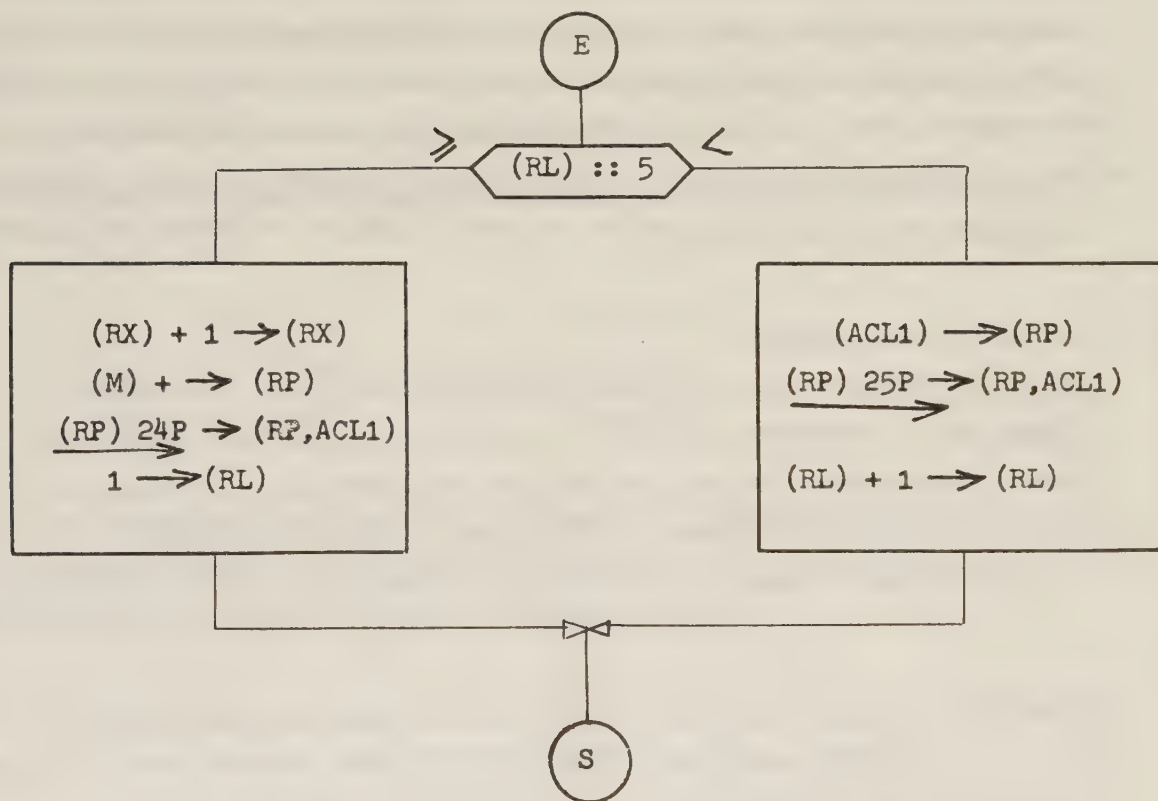
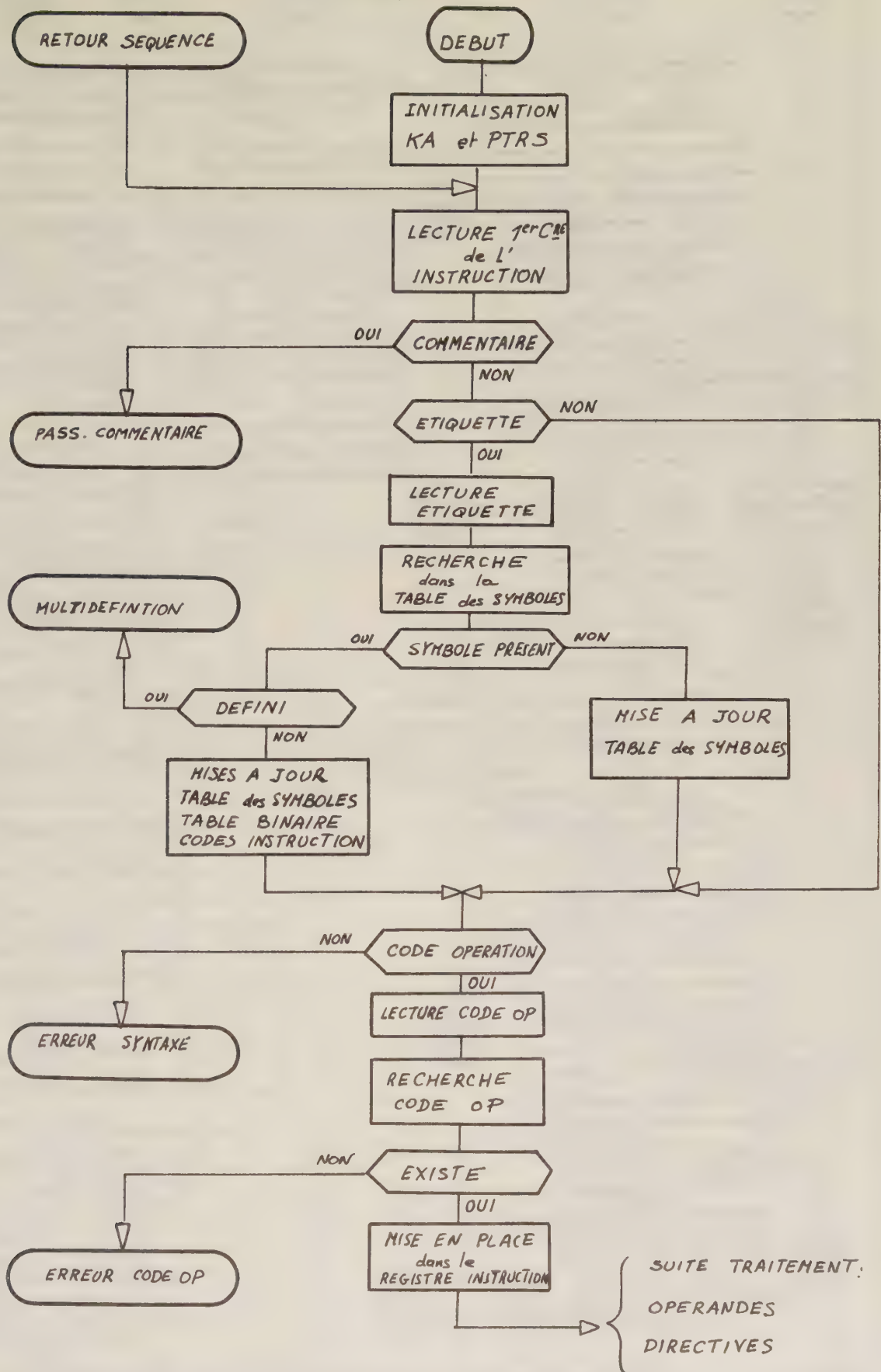
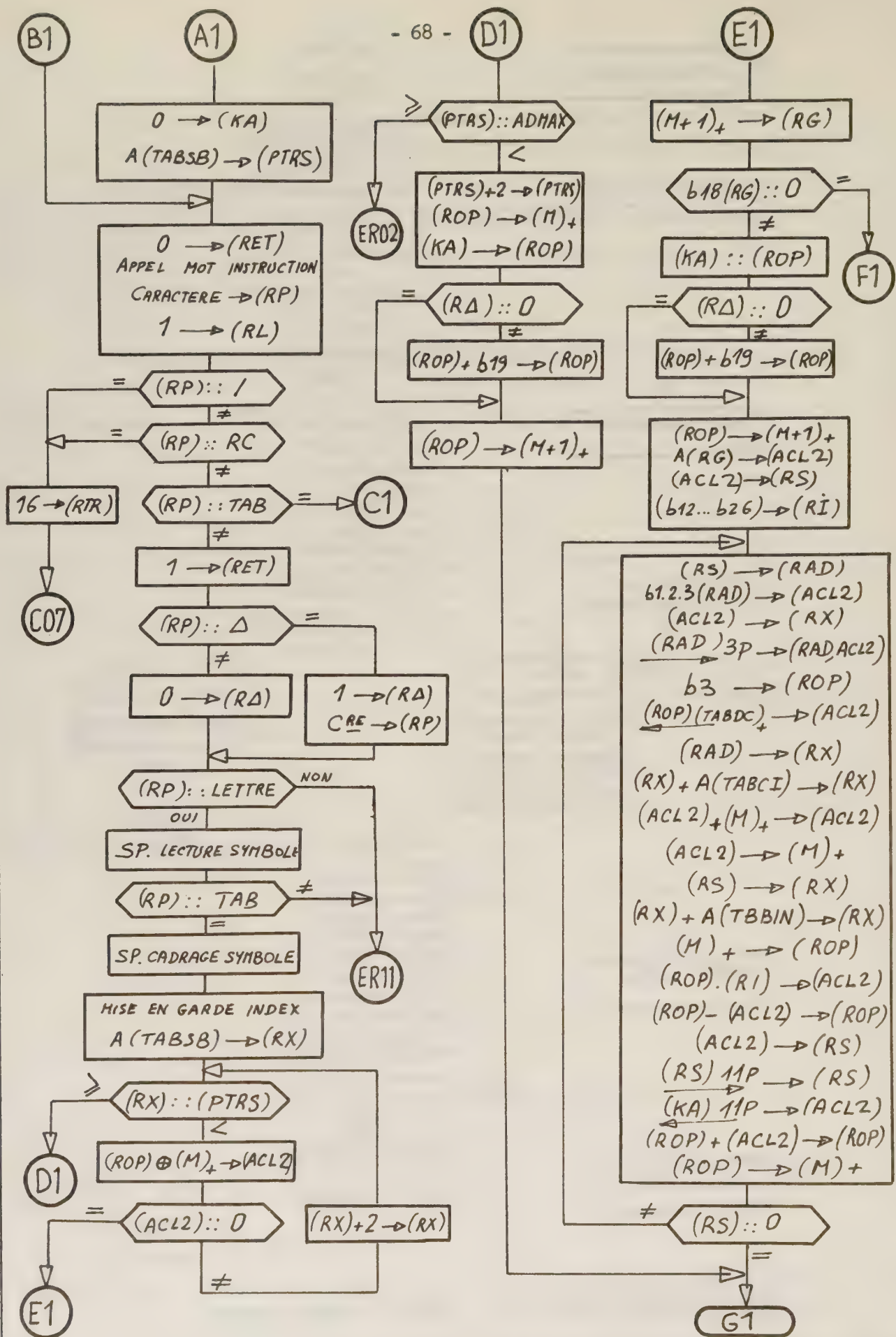
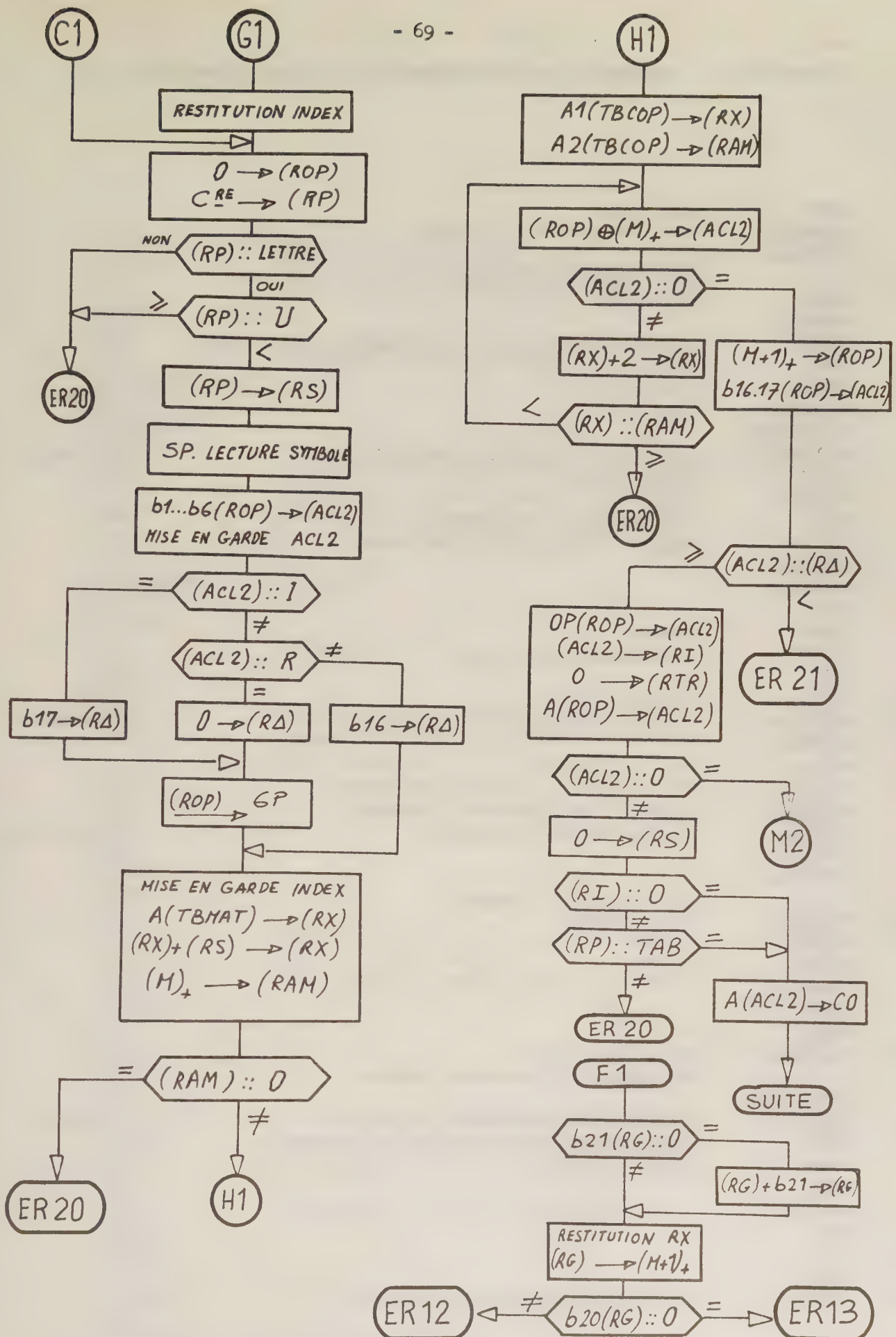


Figure 11



4_ASSEMBLEUR P1_ ORG. Fonctionnel





ASSEMBLEUR P1 : TRAITEMENT DES ETIQUETTES ET DES CODES OPERATION

0	A2EV0 A11EM31Δ A15EM3ΔS	RAZ (KA) INITIALISATION PTRS	
3	X	1ère ADRESSE TABPR	
4	A9EV0 A(EM0)+ COTV24 A3EV1 VQV25 VQV16Y14 A10EV16Z14 RV714ΔZ14 VQV30 RV92ΔZ14 A9EV1 VQV17 A4EV0Y14 RV22ΔY14 A4EV1 T13R0 COTV25 A3V1	RAZ RET 1er CARACTERE MOT PROGRAMME EN B1 (RP) TEST "RC" TEST / A(CO7) - PASSAGE COMMENTAIRE TEST "TAB" A(C1) TRAITEMENT ETIQUETTE TEST (ETIQUETTE EXTERNE) RP=RØ CARACTERE EN B1 (RP)	(466 Δ P2)
22	IQV32 RV958ΔY14 A12EV26Δ RV121Δ	TEST LETTRE A(ER11) ETIQUETTE ILLEGALE	(50 Δ P3)
26	VQV30 RV958ΔY14 RCV648Δ T15R10 A15EM31ΔS	Bt SP SYMBOLE TEST "TAB" (SEPARATION ZONE) A(ER11) ETIQUETTE ILLEGALE A(SP CADRAGE SYMBOLE)	(400 Δ P2)
31	X	1ère ADRESSE TABSB	
32	V15QR11 RV38ΔZ14 V1QM0+ RV49ΔZ14 A15V2 RV32Δ	TEST INDEX ROP=R1	
38	V11QM39ΔS X T10ER15Z14 RV920ΔZ14 A11V2 T1M0+ T2R1 RV47ΔZ4 A1W2	TEST POINTEUR DERNIERE +1 ADRESSE TABSB A(ERO2) - DEBORDEMENT TABSB MISE EN PLACE SYMBOLE ET ADRESSE DANS LA TABLE	(2 Δ P3)
47	T1M1+ RV91Δ		
49	A7EM1+	TRAITEMENT SYMBOLE EXISTANT	

ASSEMBLEUR P1

	I7QW1	TEST SYMBOLE DEFINI	
	RV83AZ14	Bt TRAITEMENT SYMBOLE MULTIDDEFINI	
	T2R1	TRAITEMENT SYMBOLE DEFINI	
	RV55AZ4		
	A1W2		
55	T1M1+	MISE EN PLACE ADRESSE DANS TABLE	
	I7QV3Z767		
	T14R5		
	A6EV129024		
	A6W511		
60	T5R8	MODIFICATION DES CODES INSTRUCTIONS	
	I8QV7	BINAIRES (CODE CHAINAGE EN CODE	
	T14R15	TRANSLATABLE)	
	C8QV3		
	A1EV4		
	K1QM1529A+	A (TABDC)	(344 Δ P4)
	T8R15		
	A15M68AS		
68	X	A (TABCI)	
	A14M0+		
	T14M0+		
	T5R15	MISE EN PLACE ADRESSE DANS LES	
	A15M73AS	INSTRUCTIONS BINAIRES (ADRESSES	
	X	DE CHAINAGE)	
	A1EM0+		
	I1QR6		
	S1R14		
	T14R5		
	K2QV11		
	A1R14		
	T1M0+		
	RV91AZ5	TEST ADRESSE DE CHAINAGE NULLE	
	RV60A	(FIN DE LA CHAINE)	
83	I7QW8	TRAITEMENT SYMBOLE MULTIDDEFINI	
	RV86AY14		
	A7W8		
	T7M1+		
	A15ER10		
	I7QW4		
	RV972AZ14	A (ER13) INTERNE MULTIDDEFINI	(54 Δ P3)
	RV970A	A (ER12) EXTERNE MULTIDDEFINI	(52 Δ P3)
91	A15ER10	TRAITEMENT CODE OPERATION	
92	A1EV0	C1	
	V3QV5	CARACTERE SUIVANT EN B1 (RP)	
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RV102AY14		
	A15V1		
	AOEM0+		

ASSEMBLEUR P1

	COTV24		
	A3EV1		
	IOQV32	TEST LETTRE	
	RV1011AZ14	A(ER20) CODE OP ILLEGAL	(93 ΔP3)
	SOV53	TEST LETTRE <U	
	RV1011ΔP0	A(ER20) CODE OP ILLEGAL	(93 ΔP3)
	TOR5	ler CARACTERE CODE OP → (RS)	
	A12EV200Δ	A(RETOUR S.P. SYMBOLE)	
108	V3QV5	S.P. LECTURE SYMBOLE (A01)	
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RV117ΔY14		
	A15V1		
	A0EM0+		
	COTV24		
	A3EV1		
	K0QV27		
	TOR14P14		
	S14V10P14		
	RR12P14	TEST FIN SYMBOLE (D01)	
121	TOR1	B01	
	V3QV5		
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RV131ΔY14		
	A15V1		
	A0EM0+		
	COTV24		
	A3EV1		
131	K0QV27		
	TOR14P14		
	S14V10P14		
	RR12P14	E01	
135	K1QV6	C01	
	T14R1		
	A1R0		
	V3QV5		
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RV147ΔY14		
	A15V1		
	A0EM0+		
	COTV24		
	A3EV1		
147	K0QV27		
	TOR14P14		
	S14V10P14		

ASSEMBLEUR P1

	RR12P14	F01	
	K1QV6		
	T14R1		
	A1R0		
	V3QV5		
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RV163AY14		
	A15V1		
	AOEMO+		
	COTV24		
	A3EV1		
163	K0QV27		
	TOR14P14		
	S14V10P14		
	RR12P14	GO1	
	K1QV6		
	T14R1		
	A1R0		
	V3QV5		
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RV179AY14		
	A15V1		
	AOEMO+		
	COTV24		
	A3EV1		
179	K0QV27		
	TOR14P14		
	S14V10P14		
	RR12P14	HO1	
	K1QV6		
	T14R1		
	A1R0		
	V3QV5		
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RV195AY14		
	A15V1		
	AOEMO+		
	COTV24		
	A3EV1		
195	K0QV27		
	TOR14P14		
	S14V10P14		
	RR12P14	IO1	
	<u>RV1(X)4Δ</u>	A (ER11.20)	(86 Δ P3)

Ce sous-programme est utilisé en sous-programme ouvert car il est très souvent appelé en cours d'assemblage; le temps d'exécution est ainsi minimisé.

c) - Translation des adresses des tables des codes opération :

Pour que le système d'assemblage puisse être placé à n'importe quel endroit de la mémoire, il est nécessaire que les adresses contenues dans les tables des codes opération soient aussi translatables.

Lors du traitement des codes opération, l'assembleur effectue la translation de la façon suivante:

- les adresses contenues dans les tables sont relatives à l'adresse 213 de la section P1.
- pour connaître l'adresse réelle de l'instruction 213, on place en 212 une instruction de branchement et retour à l'adresse 213. Du point de vue programme cela ne change rien, mais cette opération a pour effet de placer la valeur actuelle du compteur ordinal dans le registre 12 : c'est l'adresse réelle de l'instruction 213.
- il suffit ensuite d'ajouter à toutes les adresses extraites des tables, le contenu de ce registre pour obtenir les adresses absolues correspondantes.

VI - TRAITEMENT DES OPERANDES

La traduction des divers opérandes et la mise en place dans le mot machine (registre instruction) est compliquée par les nombreux cas particuliers du langage machine CAB500 surtout pour les instructions d'entrée-sortie. Suivant les limites sur les opérandes et les possibilités d'adresse pour l'opérande 1, il est possible de définir 14 classes principales (cf. annexe n° 3).

1 - Orientation du traitement

Le cas des instructions sans opérande étant réglé dans la partie traitement des codes opération, le branchement aux diverses entrées s'obtient directement par l'adresse contenue dans la table des codes opération (cf. paragraphe précédent). Il n'est pas nécessaire de prévoir 14 sections de programme traitant chacune une classe déterminée, il est plus élégant de partir de segments d'ossature suffisamment large dans lesquels le traitement est orienté par des indicateurs.

Finalement il reste 5 sections principales :

- instructions normales à deux opérandes : classes 1-2-3-4-5
- instructions de branchement : classe 6
- instructions d'entrée : classes 7-8-9
- instruction de sortie alphanumérique : classe 10
- instructions de sortie de nombres : classes 11-12-13

Toutes ces sections se regroupent pour le traitement commun de fin d'instruction (test des modificateurs, présence de commentaire, mise en place des codes instruction et des instructions binaires dans la table).

Deux registres sont utilisés dans cette optique :

a) - registre clé instruction (RCI) :

cet indicateur est typique de l'instruction à assembler, il indique en cours de traitement les branchements et les tests à effectuer sur les limites et les interdictions des opérandes.

Par exemple, si l'indicateur RCI est positionné à 0 au début de traitement cela signifie :

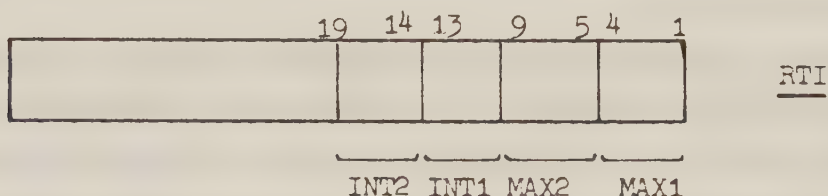
- pas de numéro de registre interdit pour le 1er opérande
- le numéro de registre interdit pour le 2ème opérande est celui du registre contenant le 1er opérande (OP1 $\neq$ OP2).

b) - registre de test instruction (RTI) :

Il contient les diverses limites et interdictions concernant les opérandes c'est-à-dire :

- MAX1 : nombre maximum désignant le 1er opérande
- MAX2 : nombre maximum désignant le 2ème opérande
- INT1 : numéro de registre interdit pour le 1er opérande
- INT2 : numéro de registre interdit pour le 2ème opérande

Ces indications sont cadrées dans le registre RTI de la façon suivante :



2 - Traduction des opérandes

Suivant les instructions nous savons que le nombre d'opérandes varie de 1 à 3, de toute façon les deuxième et troisième opérandes sont des nombres décimaux de deux chiffres, au maximum et ils peuvent être traités comme un numéro de registre.

Le premier opérande pouvant être un registre, une mémoire ou immédiat le branchement à la séquence correspondante est réalisé par le test du contenu du registre R Δ , positionné au cours du programme P1 par la lecture du dernier caractère constituant le code opération. Voyons les problèmes posés par chaque cas :

- a) - Opérande en registre : (cf. SP REGISTRE : SP2) l'opérande est défini par un nombre décimal comportant au plus deux chiffres. Pour la translittération on fait appel à un sous-programme qui prévoit la lecture de trois caractères au plus (2 chiffres et un caractère de séparation).

En sortie du sous-programme, on vérifie le cliché de l'instruction (premier opérande et deuxième opérande sont séparés par une virgule) et la valeur du nombre par rapport au maximum admissible et aux cas d'interdiction.

b) - Opérande immédiat : (cf. SP IMMEDIAT : SP3)

- dans le cas général (RCI = 0) il suffit de reconnaître la présence ou l'absence du facteur d'échelle "X" qui détermine le code d'adresse 1 correspondant et la valeur maximum admissible.

Le test du premier caractère oriente vers la lecture décimale (SP4) ou hexadécimale (SP5) qui est suivie des tests de légalité d'opérande habituels.

- dans le cas des opérations de décalage (RCI $\neq$ 0), le nombre de positions est spécifié dans l'instruction symbolique par un nombre N compris entre 0 et 32 alors que dans le langage machine c'est un digit placé dans une position N' de l'instruction, tel que :

$$N' = N + 1 \quad \text{pour les décalages à gauche}$$

$$N' = 33 - N \quad \text{pour les décalages à droite}$$

(cf. annexe n° 5).

Le code d'adresse 1 et la valeur immédiate du digit de décalage étant définis de la façon suivante :

$$\begin{array}{lll} \text{Si } N' \leq 17 & \text{CAD1} = 4 & \text{valeur } V = 2^{N'} \\ \text{Si } N' \geq 18 & \text{CAD1} = 6 & \text{valeur } W = 2^{N'-17} \end{array}$$

Le problème peut se traiter par modification d'instruction, mais cela est dangereux surtout en cours de mise au point, il est bien plus simple de faire un appel indexé à une table des 2^N ; la valeur correcte de l'index est calculée à partir de N et de la nature du décalage (cf. SP3 paragraphe X).

c) - Opérande symbolique : (cf. SP EXPRESSION SYMBOLIQUE : SP6)

Nous avons déjà vu que l'opérande peut être une expression arithmétique (opérateurs + et -) de symboles et constantes.

Le sous-programme doit calculer l'adresse correspondant à l'expression et vérifier les points suivants :

- l'expression est effectivement calculable, tous les symboles en étant définis ; sinon, le symbole non défini ou externe est utilisé seul.
- translatabilité et validité de l'expression et mise en place d'un code rappelant la nature de l'adresse qui définit l'opérande : absolue, translatable ou de chaîne.

Il utilise pour cela les sous-programmes : SP décimal et SP hexadécimal pour le calcul des constantes et le SP SYMBOLE (SP1) pour les facteurs symboliques. Dans tous les cas, l'adresse correspondante est placée dans un registre déterminé qui est additionné ou soustrait (suivant le signe qui précède le terme) d'un registre totalisateur.

En fin d'expression ce registre contient l'adresse voulue, si celle-ci est négative le système en prend la valeur absolue.

Pour les détails du traitement consulter le SP6 (paragraphe X-6).

3 - Tests de fin d'instruction

Après traduction des opérandes, la présence d'une virgule indique que l'instruction est complétée par l'un des modificateurs : indexage (+) ou indice de saut (S).

Ils n'ont pas nécessairement de sens dans toutes les instructions, ainsi : l'indice de saut est incompatible avec l'instruction d'écriture alphanumérique et toutes les opérations microprogrammées.

L'indexage n'est possible qu'avec les instructions désignant un opérande en mémoire.

Pour réaliser cela, la valeur du registre RC est testée ((RC) = 0 pour une instruction d'écriture alphanumérique, (RC) = 1 dans tous les autres cas) ainsi que le digit B27 du registre d'instruction (B27 = 1 indique que l'opération fait appel à un microprogramme). Si (RC) = 0 ou B27 (RI) = 1, le caractère analysé ne peut être que le modificateur d'indexage sinon le programme se débranche à la section d'élaboration du diagnostic d'erreur correspondant. Dans l'autre cas, l'instruction à translitérer est quelconque, les tests du code d'adresse 1 (digits B2, B3 et B4 du registre d'instruction) permettent de passer directement à la comparaison au modificateur S pour une instruction de lecture alphanumérique (CAD1 = 1) ; si ce code d'adresse est différent de 1, il doit être celui d'un opérande en mémoire (CAD2 = 2) sinon la syntaxe de l'instruction n'est pas respectée (débranchement "erreur").

Après ces différents tests la traduction de l'instruction est normalement finie, la commande est alors passée à un sous-programme de traitement de fin d'instruction qui réalise les opérations suivantes :

- vérification du cliché de l'instruction : retour chariot de fin d'instruction ou tabulation indiquant la présence d'un commentaire à ignorer.
- mise en place de l'instruction (ou des constantes dans le cas d'utilisation du sous-programme à partir d'une directive).
- mise en place d'un drapeau d'erreur si l'instruction comporte au moins un diagnostic.
- génération du code d'instruction binaire (utilisé pour le "listing" et le chargement des programmes) à partir de la valeur du registre RTR.
- évolution du compteur d'adresse et de l'index du programme source pour le passage à l'instruction suivante.

- test de fin de programme : normalement le programme source n'est pas terminé et le branchement au traitement des étiquettes est effectué (segment P1 repère B1), dans le cas contraire il y a émission d'un diagnostic car le programme doit obligatoirement se terminer par une directive "FIN".

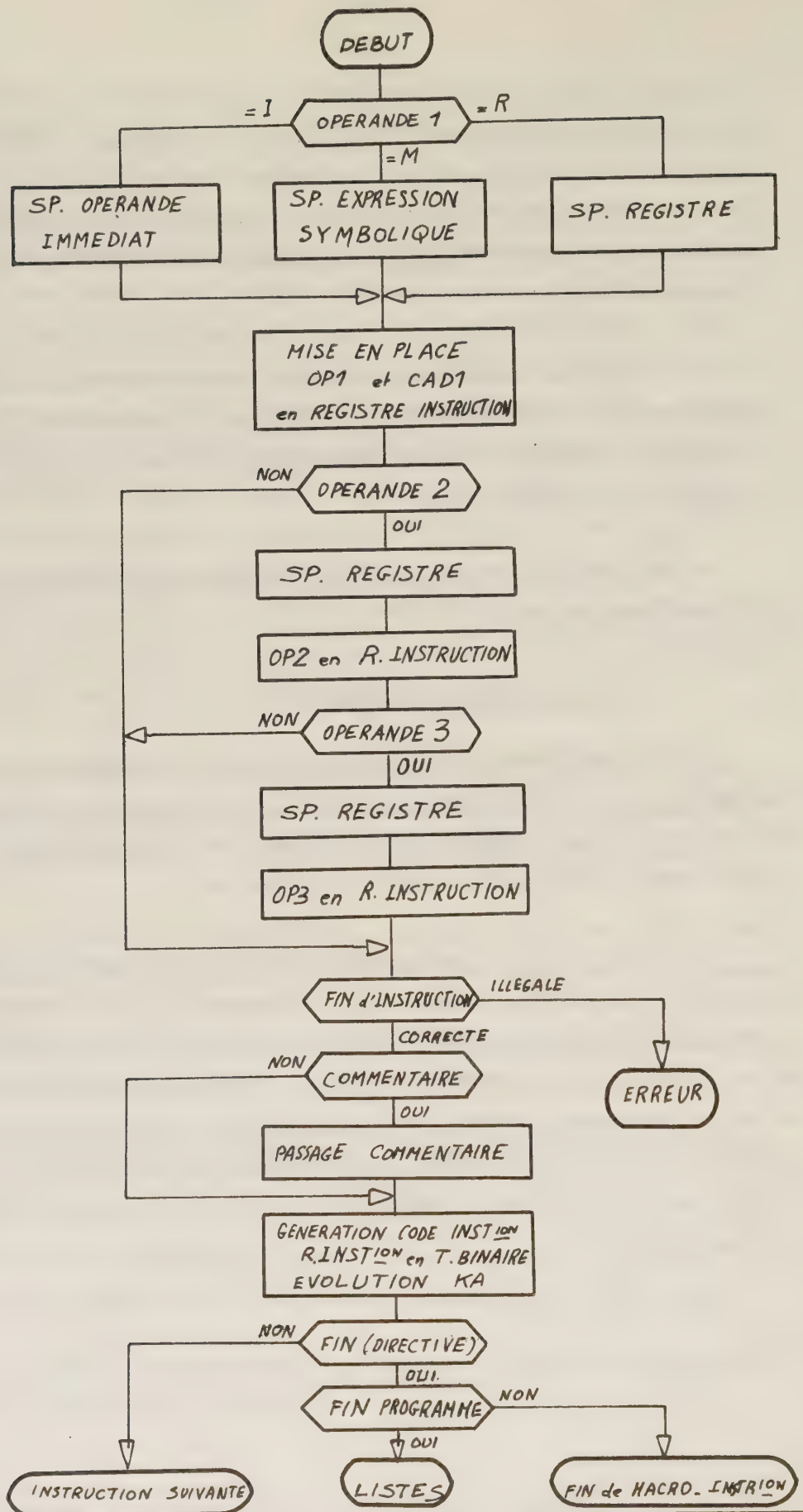
C'est dans la séquence traitant cette directive que se situe le branchement à la 2ème phase de l'assemblage, c'est-à-dire l'éclatement et le rangement alphabétique des tables des symboles ainsi que les éditions éventuelles de listes.

Le détail du traitement est fourni dans le sous-programme n° 7 (paragraphe X-7).

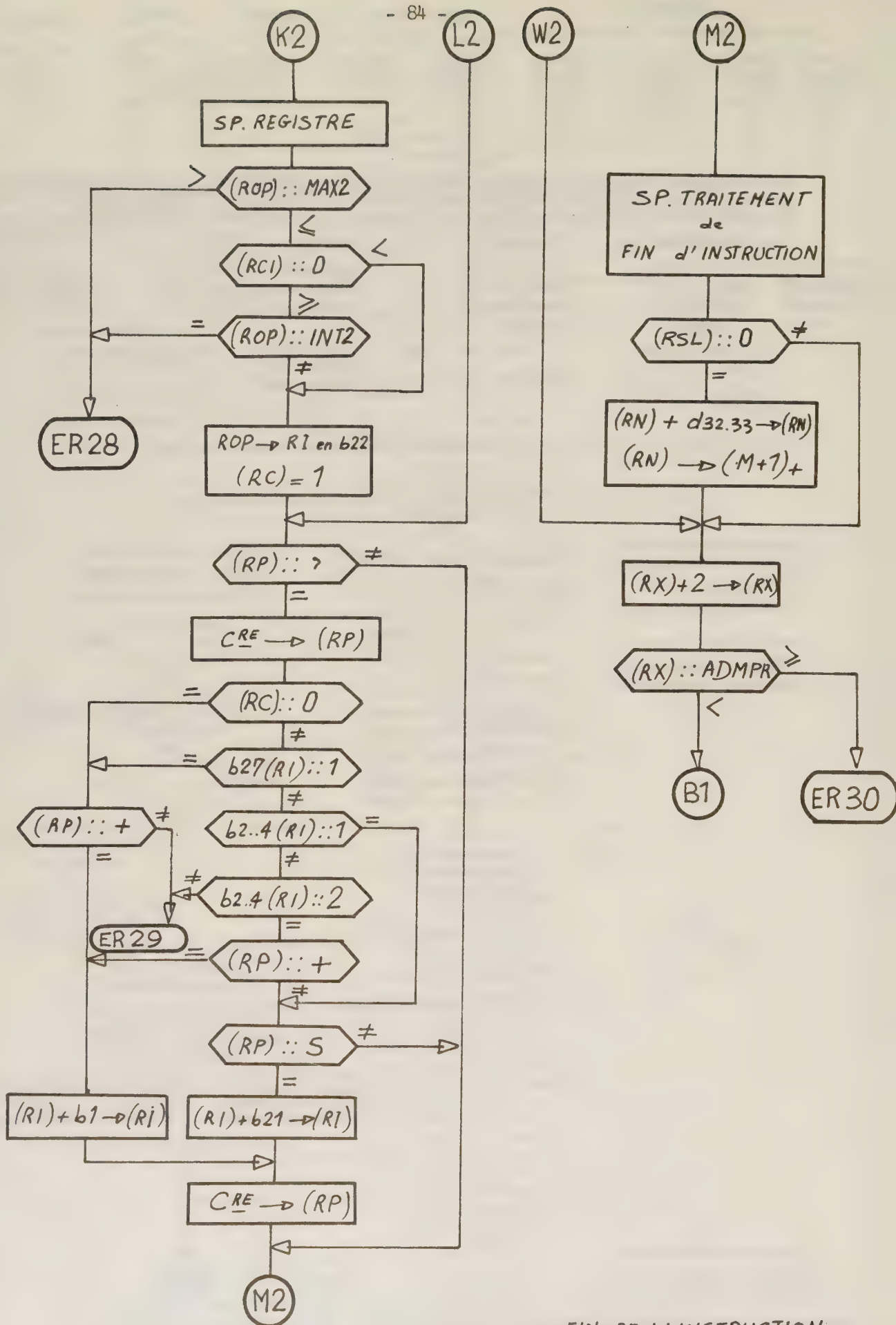
4 - Organigrammes et programmes

L'organigramme simplifié représente un traitement idéalisé des opérandes d'une instruction quelconque, c'est dans les organigrammes machine qu'il faut chercher le traitement réel correspondant à chacune des cinq classes d'instructions définies plus haut.

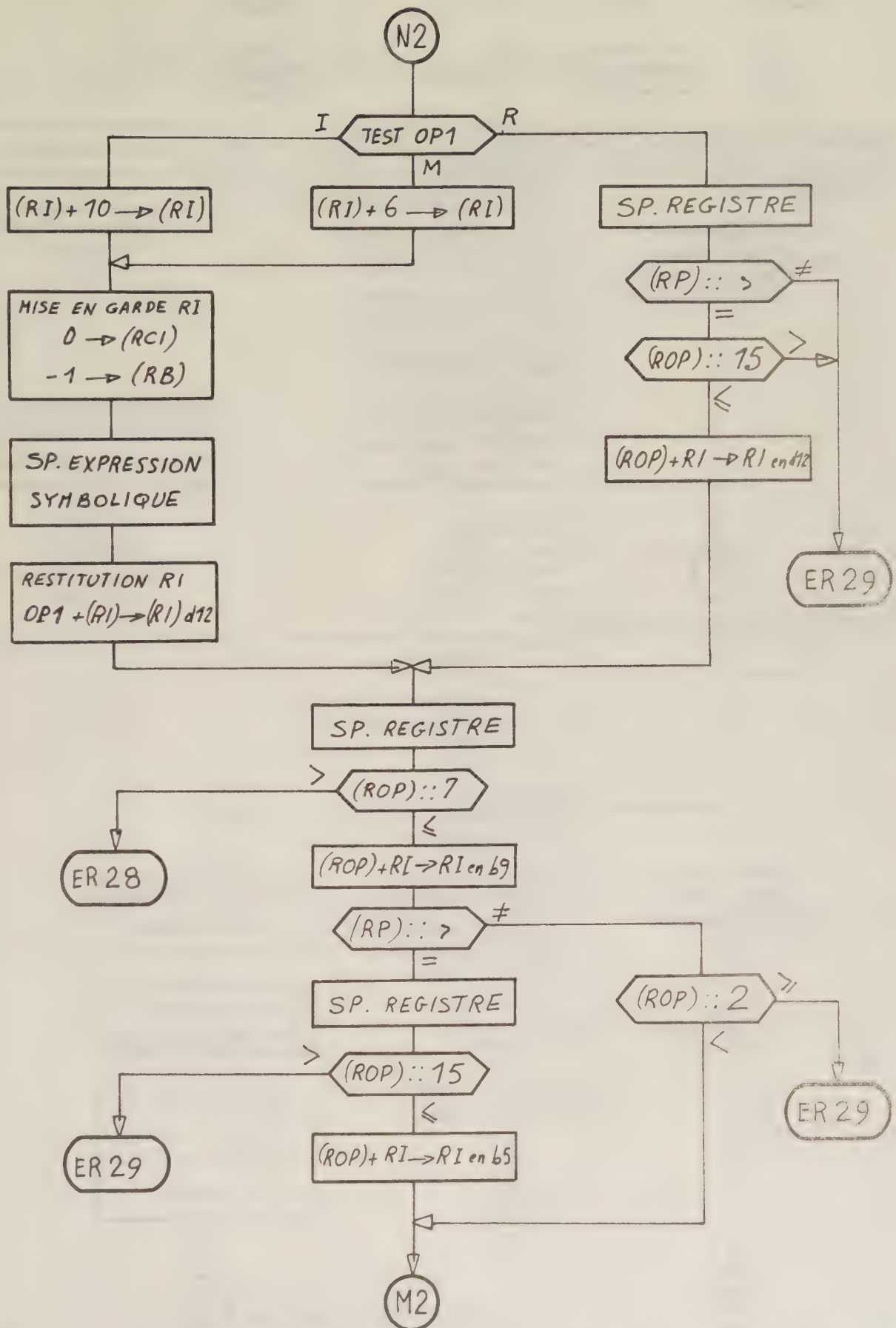
Du point de vue programmation en langage autocode CAB500, les sections "instructions normales à 2 opérandes" et "fin d'instruction" représentent 80% du segment d'assembleur P2. Ils comprennent en effet la majorité des sous-programmes utilisés à l'assemblage, ceux-ci sont employés en sous-programmes ouverts dans cette séquence et en sous-programmes fermés dans toutes les autres. Les méthodes de liaisons et d'appels utilisées sont définies au paragraphe traitant des sous-programmes (paragraphe X-1).



7 - ASSEMBLEUR P2 - ORG. Simplifié

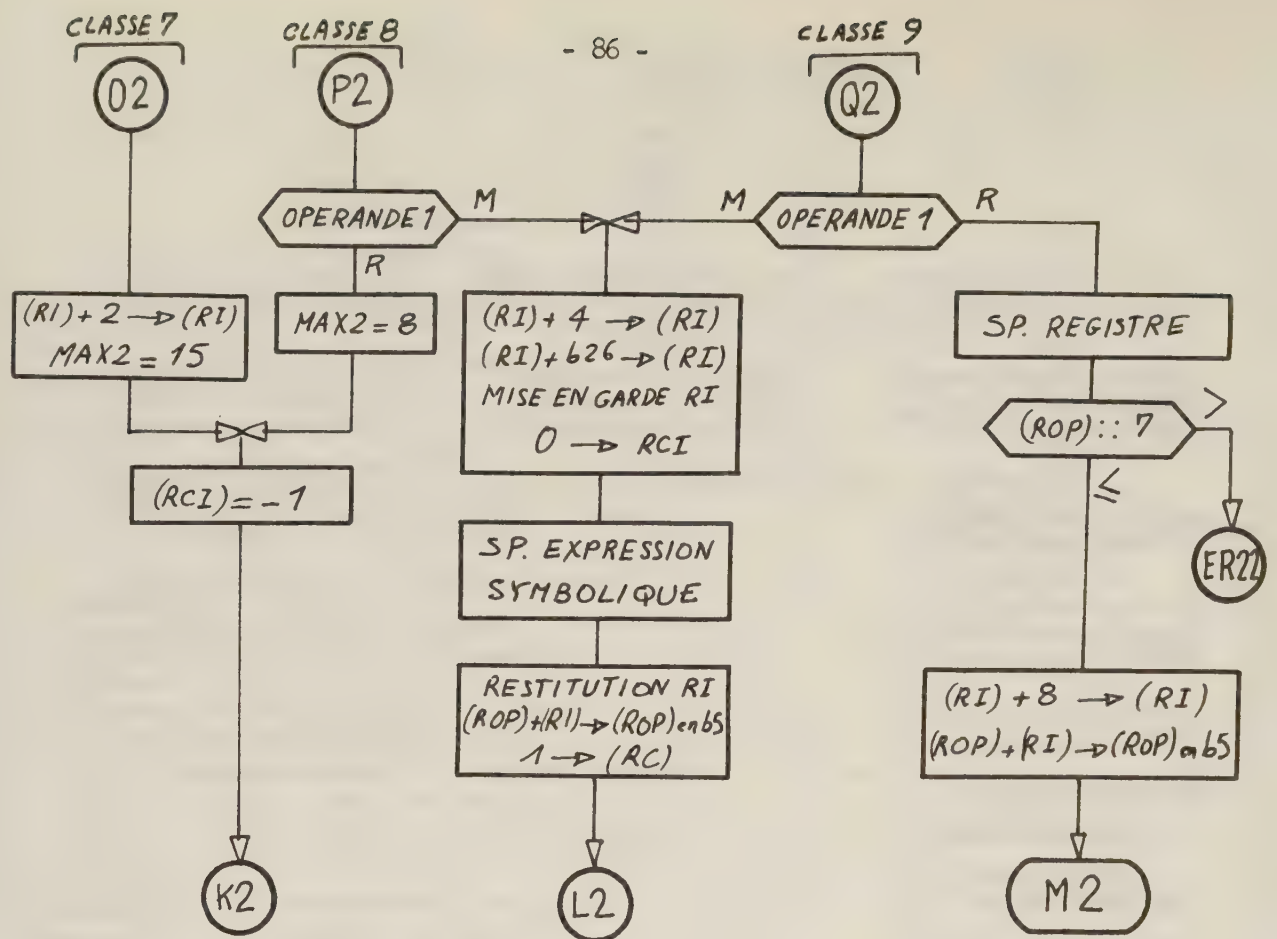


- FIN DE L'INSTRUCTION -

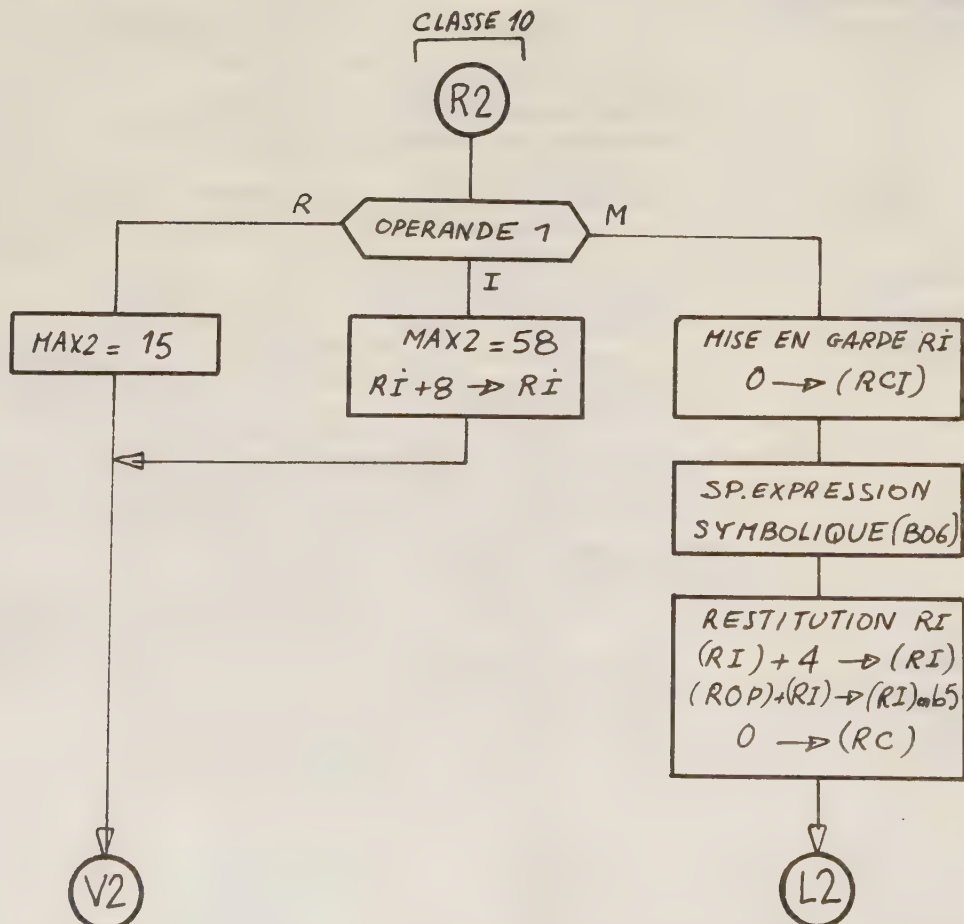


INSTRUCTIONS de BRANCHEMENT -

10 - ASSEMBLEUR P2 - ORG. Machine (3)

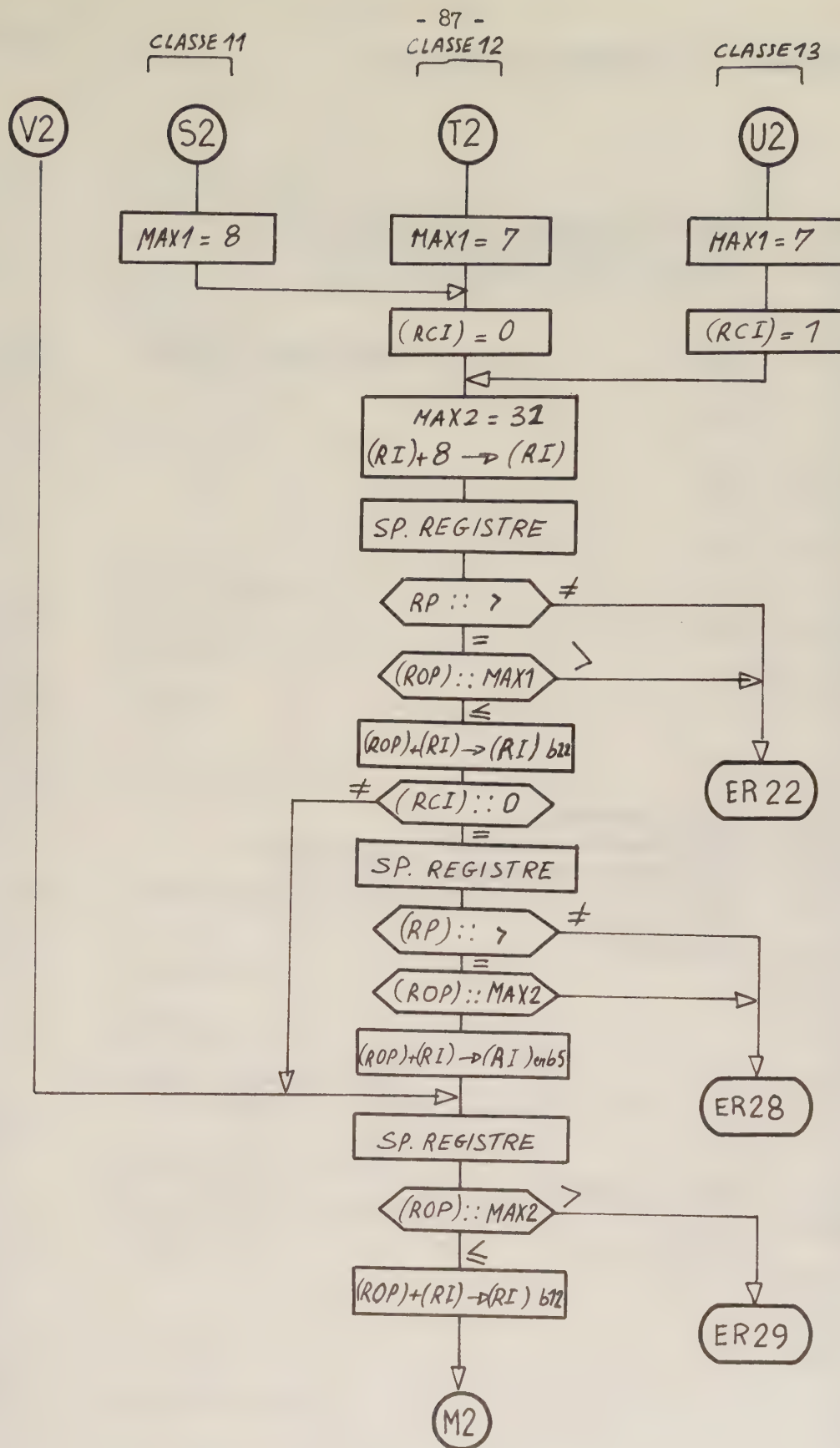


INSTRUCTIONS d'ENTREE -



INSTRUCTION de SORTIE (Ecriture ALPHANUMERIQUE) -

11 - ASSEMBLEUR P2 - ORG. Machine (4)



INSTRUCTIONS de SORTIE (NOMBRES) -

12 - ASSEMBLEUR P2 - ORG. Machine (5)

ASSEMBLEUR P2 : TRAITEMENT DES OPERANDES

0	A7EV1	MISE EN PLACE INDICATEURS INT1,
	RV3Δ	INT2, MAX1, MAX2 EN REGISTRE
	S7EV2	RTI ET CLE EN REGISTRE RCI
3	A5V119	
	RV22Δ	
	A5V248	G2 MISE EN PLACE INDICATEURS
	RV8Δ	
	A5V136	F2
8	S7EV1	
	RV22Δ	
	A5V6656	E2
	RV13Δ	
	A5V7168	D2
13	A7EV3	
	RV21Δ	
	A5V6656	C2
	RV18Δ	
	A5V7168	B2
18	A7EV2	
	RV21Δ	
	A7EV0	A2
21	A5V255	
22	S4V32768	TEST OPERANDE 1
	RV239ΔZ4	TRAITEMENT OPERANDE SYMBOLIQUE
	RV103ΔP4	TRAITEMENT OPERANDE IMMEDIAT
	A12EV65Δ	A (SORTIE SP REGISTRE) EN REGISTRE 12
26	V3QV5	SP REGISTRE
	T13R1Y14	LECTURE C1
	C1TV25Y14	
	A3V1Y14	
	RV35ΔY14	
	A15V1	
	A1EM0+	
	C1TV24	
	A3EV1	
35	T1R14	
	S14V10	TEST CHIFFRE
	RV767ΔP14	A (ER22)
	V3QV5	LECTURE C2
	T13R0Y14	
	C0TV25Y14	
	A3V1Y14	
	RV47ΔY14	
	A15V1	
	A0EM0+	
	C0TV24	
	A3EV1	
47	T0R14	
	S14V10	TEST CHIFFRE
	RR12P14	SORTIE 1

(97Δ P3)

ASSEMBLEUR P2

	K1QV3	CALCUL C2+10C1	
	AOR14		
	K1QV1		
	AOR14		
	TOR1	ROP=R1	
	V3QV5	LECTURE C3	
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RR12Y14	SORTIE 2	
	A15V1		
	A(EM)+		
	COTV24		
	A3EV1		
	RR12	SORTIE 3	
65	VQV12	TEST VIRGULE	
	RV767ΔY14	A (ER22)	(97Δ P3)
	I5QV15	TEST MAX1	
	S14R1		
	RV767ΔN14	A (ER22)	(97Δ P3)
	RV81ΔN7		
	RV79ΔZ7		
	I5QV7680	TEST OP1=INT1	
	T14R9		
	C9QV9		
	S9R1		
	RV767ΔZ9	A (ER22)	(97Δ P3)
	A9R1		
	RV81ΔZ9		
79	K1QV13	INT2=(ROP)	
	A5R14		
81	T7R9	TEST RCI=-2	
	A9V2		
	K1QV4	(ROP) CADRE EN B5	
	A6V8Z9	CODE ADRESSE 1=B4 POUR INSTRUCTIONS D.L. (RCI=-2)	
	RV87ΔZ9		
	K1QV11	(ROP) CADRE EN B12	
87	A6R14	OP1 DANS (RI)	
	RCV26Δ	TRAITEMENT OPERANDE 2 (K2)	
89	I5QV496	TEST MAX2	
	T14R9		
	C9QV4		
	S9R1		
	RV779ΔN9	A (ER28)	(109Δ P3)
	RV99ΔN7		
	C5QV13	TEST OP2=INT2	
	I5QV15		
	S14R1		
	RV779ΔZ14	A (ER28)	(109Δ P3)
99	K1QV21	(ROP) CADRE EN B22	

ASSEMBLEUR P2

	A6R14	OP2 DANS (RI)
	A8EV1	(RC)=1
	RV424Δ	VERS TESTS ZONE OPERANDE 3
103	A12EV163Δ	TRAITEMENT OPERANDE IMMEDIAT A (SORTIE)=163 Δ
	V3QV5	LECTURE C1
	T13ROY14	
	COTV25Y14	
	A3V1Y14	
	RV113ΔY14	
	A15V1	
	A0EM0+	
	COTV24	
	A3EV1	
113	A4EV33Y7	TEST RCI ≠ 0
	RV131ΔY7	
	V0QV13	TEST X (FACTEUR D'ECHELLE 2 ¹⁷)
	RV120ΔZ14	
	A6V8	CODE ADRESSE 1=V
	A4EV131071	MAXI=2 ¹⁷
	RV131Δ	
120	A6V12	CODE ADRESSE 1=W
	A4EV65535	MAXI=2 ¹⁶
	V3QV5	LECTURE C2
	T13ROY14	
	COTV25Y14	
	A3V1Y14	
	RV131ΔY14	
	A15V1	
	A0EM0+	
	COTV24	
	A3EV1	
131	T12EM132ΔS	MISE EN GARDE ADRESSE SORTIE SP
132	X	A (SORTIE SP)
	T0R14	TEST CHIFFRE DECIMAL
	S14V10	
	A12EV157Δ	ADRESSE RETOUR = 157 Δ
	RV138ΔN14	VERS SP DECIMAL
	RV203Δ	VERS SP HEXADECIMAL
138	T0R1	SP DECIMAL (AO4)
	V3QV5	LECTURE Ci
	T13ROY14	
	COTV25Y14	
	A3V1Y14	
	RV148ΔY14	
	A15V1	
	A0EM0+	
	COTV24	
	A3EV1	
148	T0R14	TEST CHIFFRE
	S14V10	

151	RR12P14	SORTIE (BO4)	
	K1QV3	CALCUL Ci+10(ROP) (CO4)	
	AOR14		
	K1QV1		
	AOR14		
	TOR1		
	RV139Δ		
157	VQV12	TEST VIRGULE	
	RV769ΔY14	A(ER23)	(99Δ P3)
	S4R1	TEST MAXI	
	RV769ΔY14	A(ER23)	(99Δ P3)
	RV167ΔY7	TEST RCI	
	RM132Δ	SORTIE SP IMMEDIAT (CO3)	
	K1QV4	(ROP) CADRE EN B5	
	A6R14	OP1 → (RI)	
	RV394Δ		
	F		
167	T15ER10	GARAGE (RX)	
	SlV16	CALCUL QUANTITE IMMEDIATE DECALAGE	
	V7QV2		
	A15R1Y14		
	S15R1Z14		
	RV175ΔZ15		
	A6V12P15		
	RV177ΔP15		
	A6V8		
	A15V17		
	AlEM182Δ+	APPEL INDEXE TAB2N	
	T10ER15	RESTITUTION INDEX	
	K1QV4	REPETITION SEQUENCE CHARGEMENT OP1	
	A6R14	EVITE PERTE D'UN TOUR TAMBOUR	
	RV394Δ		
	F	TABLE DES PUISSANCES DE 2	
183	1		
	2		
	4		
	8		
	16		
	32		
	64		
	128		
	256		
	512		
	1024		
	2048		
	4096		
	8192		
	16384		
	32768		
	65536		

ASSEMBLEUR P2

	X	RESERVE	
	X		
	F		
203	V0QV56	TEST X TRAITEMENT HEXADECIMAL	
	RV769ΔY14	A (ER23)	(99Δ P3)
	V3QV5	LECTURE	
	T13ROY14		
	C0TV25Y14		
	A3V1Y14		
	RV214ΔY14		
	A15V1		
	A0EM0+		
	C0TV24		
	A3EV1		
214	V0QV14	TEST "="	
	RV769ΔY14	A (ER23)	(99Δ P3)
216	A6EVO	(RC)=0..ENTREE SP HEXADECIMAL (AO5)	
217	V3QV5	LECTURE Ci	
	T13ROY14		
	C0TV25Y14		
	A3V1Y14		
	RV226ΔY14		
	A15V1		
	A0EM0+		
	C0TV24		
	A3EV1		
226	TOR14	TEST CHIFFRE HEXADECIMAL	
	S14V10		
	RV234ΔN14		
	S14V23		
	RR12N14	SORTIE (BO5)	
	S14V6	CALCUL EQUIVALENT BINAIRE LETTRES A,B,C,D,E,F	
	RV769ΔP14		
	S0V23		
234	K1QV4Y8	CALCUL 16(ROP)+Ci → (ROP)	
	A0R14Y8		
	TOR1		
	A8V1		
	RV217Δ		
239	A12EV385Δ	TRAITEMENT EXPRESSION SYMBOLIQUE A(SORTIE)=385Δ	
	T6EM386Δ	GARAGE (RI) et RAZ	
	A6V1	1 → (RB)	
	T7EM392Δ	GARAGE RCI	
243	T5EM383Δ	GARAGE RTI - ENTREE (BO6)	
	A9EVO	RAZ REXT	
	A10EVO	RAZ RTR	
	T12EM247ΔS	MISE EN GARDE ADRESSE SORTIE	
247	X	A(SORTIE)	
	V3QV5	LECTURE 1er CARACTERE	
	T13R1Y14		

ASSEMBLEUR P2

	C1TV25Y14		
	A3V1Y14		
	RV257ΔY14		
	A15V1		
	A1EM0+		
	C1TV24		
	A3EV1		
257	V1QV11	TEST SIGNE -	
	A4EV0Z14	O → (RSG)	
	RV263ΔZ14		
	S4EV1		
	V1QV10	TEST SIGNE +	
	RV272ΔY14		
263	V3QV5	LECTURE C'1 (C06)	
	T13R1Y14		
	C1TV25Y14		
	A3V1Y14		
	RV272ΔY14		
	A15V1		
	A1EM0+		
	C1TV24		
	A3EV1		
272	V3QV5	LECTURE C2	
	T13ROY14		
	C0TV25Y14		
	A3V1Y14		
	RV281ΔY14		
	A15V1		
	A0EM0+		
	C0TV24		
	A3EV1		
281	T1R14	TEST CHIFFRE C1	
	S14V10		
	RV293ΔP14	VERS TESTS "SYMBOLE"	
	T0R14	TEST CHIFFRE C2	
	S14V10		
	RCV151ΔN14	VERS SP DECIMAL ET RETOUR	
287	S14V3	TEST CARACTERE SEPARATION (+-,)	
	RV290ΔZ6		
	RV771ΔP14	A (ER24)	(101 Δ P3)
290	V9QV1	TEST REXT	
	A9V1Y14		
	RV360Δ		
293	S14V7	TEST "Δ"	
	RV301ΔY14		
	T0R14	TEST CARACTERE SEPARATION (+-,)	
	S14V13		
	RV299ΔN6		
	RV771ΔP14	A (ER24)	(101 Δ P3)
299	T2R1	(KA) → (ROP)	

ASSEMBLEUR P2

301	RV355Δ S14V6 RV771ΔN14 V1QV56 RV309ΔY14 V0QV14 RV309ΔY14 A12EV287Δ RV216Δ	TEST LETTRE A (ER24) TEST "X" TEST "=" ADRESSE RETOUR SP = 287 BRANCHEMENT SP HEXADECIMAL	(101 Δ P3)
309	K0QV27 TOR14P14 S14V10P14 RCV(X)N14 TOR14 S14V13 RV318ΔN14 RV318ΔZ6 RV771Δ	TRAITEMENT OPERANDE SYMBOLIQUE TEST CARACTERE DE SEPARATION (+-, TAB RC) A (SP SYMBOLE COL) TEST CARACTERE DE SEPARATION (+-,)	(135 Δ P1)
318	RCV4(X)Δ T15ER5 A15EM321ΔS	A (ER24) A (SP CADRAGE SYMBOLE) GARAGE RX	(101 Δ P3)
321	X	A (TABSB)	
322	V15QR11 RV341ΔZ14 V1QM0+ RV328ΔZ14 A15V2 RV32Δ	TEST INDEX VERS TRAITEMENT SYMBOLE NON DEFINI TEST ROP (DISJONCTION INDEXEE) VERS TRAITEMENT SYMBOLE EXISTANT	
328	A1EM1+ I1QW5 RV334ΔY14 I1QV32767 T14R1 RV355Δ	SYMBOLE EXISTANT - APPEL MEMOIRE ADRESSE TEST ZONE INDICATEUR VERS TRAITEMENT SYMBOLE EXTERNE EXTRACTION PARTIE ADRESSE → (ROP)	
334	I1QV32767 RV338ΔZ14 A7ER14 S1R14	A (GO6) EXTRACTION PARTIE ADRESSE → (ROP) TEST ADRESSE CHAINE NULLE RAZ PARTIE ADRESSE DE (ROP) ET MISE EN PLACE DANS (REXP)	
338	A1R2 T1EM1+ RV352Δ	(KA) → PARTIE ADRESSE (ROP) TRANSFERT INDEXE (TABLE DES SYMBOLES)	
341	V11QM342ΔS	SYMBOLE NON DEFINI - TEST PTRS	
342	X T5ER15Z14 RV672ΔZ14 A11V2 T1M0+ T2R1 A1W1 RV351ΔN6	ADRESSE MAX +1 TABLE DES SYMBOLES RESTITUTION INDEX A (ERO2) DEBORDEMENT TABLE DES SYMBOLES MISE A JOUR POINTEUR MISE EN PLACE SYMBOLE ; TRANSFERT INDEXE ADRESSE (KA) ET INDICATEURS D18 : NON DEFINI (BRANCHEMENT)	(2 Δ P3)

ASSEMBLEUR P2

	A1W3	B20 : EXTERNE (INSTRUCTION QUELCONQUE)	
351	T1EM1+		
352	S10EV2	(RTR)=-2	
	S9V1	(REXT)-1 (REXT)	
	RV357Δ		
355	V9QV1	TEST (REXT)	
	A9V1Y14		
357	T5R15Y5	RESTITUTION RX	
	A10V1N4	VALEUR CODE TRANSLATABILITE	
	S10V1Z4		
360	V9QV1	TEST (REXT) (HO6)	
	RV365ΔZ14		
	S14ER9		
	V14QV1		
	RV773ΔY14	A (ER25)	(103 Δ P3)
365	A7R1N4	(REXP)+(ROP) SI PLUS (RSG > 0)	
	S7R1Z4	(REXP)-(ROP) SI MOINS (RSG = 0)	
	T0R4	TEST CARACTERE SEPARATION (+-,)	
	S4V11		
	RV263ΔN4	NOUVEAU TERME A ANALYSER A(CO6)	
	RV263ΔZ4		
	A14ER7P7	EXPRESSION TERMINEE	
	S14ER7N7		
	T14R7N7	SI (REXP) < 0 - (REXP) → (REXP)	
	S14M375ΔS	TEST VALEUR MAXIMUM	
375	X	ADRESSE MAXIMALE	
	RV777ΔP14	A (ER27)	(107 Δ P3)
	RV382ΔZ10	TEST VALIDITE TRANSLATABILITE (RTR)	
	A9ER10	0 NON TRANSLATABLE	
	S9ER10N10	1 TRANSLATABLE	
	V9QV1	-1 CHAINAGE	
	RV775ΔY14	A (ER26)	(105 Δ P3)
382	A5EM383ΔS	RESTITUTION RTI	
383	X	GARAGE RTI	
	RM247Δ	SORTIE SP TRAITEMENT EXPRESSION	
	A6EM386ΔS	RESTITUTION RI	
386	X	GARAGE RI	
	A6V4	(RI)+4 → (RI) (CODE ADR1)	
	X	PLACE RESERVEE	
	K7QV4		
	A6R14	OP1 DANS RI EN B5	
	A7EM392ΔS	RESTITUTION RCI	
392	X	GARAGE RCI	
	A8EV1	1 → (RC)	
394	V7QV1		
	RV397ΔZ14		
	S7EV1		
397	A12EV89Δ	ADRESSE RETOUR = 89Δ (K2)	
	RV26Δ	VERS SP REGISTRE (TRAITEMENT 2ème OPERANDE)	
	F	PLACE RESERVEE	

ASSEMBLEUR P2

400	ILQW4096	SP CADRAGE SYMBOLE (AO8)	
	RR12Y14	SORTIE 1 (BO8)	
	K1QV6	(ROP) + ESPACE → (ROP)	
	A14V15		
	T14R1		
	ILQW4096		
	RR12Y14	SORTIE 2 (CO8)	
	K1QV6		
	A14V15		
	T14R1		
	ILQW4096		
	RR12Y14	SORTIE 3 (DO8)	
	K1QV6		
	A14V15		
	T14R1		
	ILQW4096		
	RR12Y14	SORTIE 4 (EO8)	
	K1QV6		
	A14V15		
	T14R1		
	RR12	SORTIE 5 (FO8)	
	X	PLACES RESERVEES	
	X		
	F		
424	VQV12	TESTS DE FIN INSTRUCTION_TEST "VIRGULE" (L2)	
	RV461ΔY14	VERS M2	
	V3QV5	LECTURE CARACTERE SUIVANT	
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RV435ΔY14		
	A15V1		
	A(OEM)+		
	COTV24		
	A3EV1		
435	RV438ΔZ8	TEST RC	
	I6QW512	TEST B27	
	RV441ΔZ14		
438	VQV10	TEST + (INDEXAGE)	
	RV447ΔZ14		
	RV781Δ	A (ER29)	(111 ∠ P3)
441	I6QV14	TESTS B2,3,4	
	S14V2		
	RV449ΔZ14		
	V14QV2		
	RV781ΔY14	A (ER29)	(111 Δ P3)
	VQV10	TEST +	
	A6V1Z14		
	RV452ΔZ14		
449	VQV51	TEST S (INDICE DE SAUT)	

	RV461AY14		
	A6W8		
452	V3QV5	LECTURE	
	T13R0Y14		
	C0TV25Y14		
	A3V1Y14		
	RV461AY14		
	A15V1		
	A0EM0+		
	C0TV24		
	A3EV1		
461	A12EV531Δ	SP FIN INSTRUCTION (AO7) - MISE EN GARDE A (RETOUR)	
462	V0QV25	TEST "RC" (ENTREE BO7)	
	RV472AZ14		
	V0QV30	TEST "TAB"	
	RV781AY14	A (ER29)	(111 Δ P3)
466	A0EM1+	PASSAGE COMMENTAIRE (ENTREE CO7)	
	RV470AN0	TEST MOT SEPARATION	
	A15V1		
	RV466Δ		
470	V10QV16	TEST RTR=16 - LIGNE DE COMMENTAIRE	
	RV534AZ14	PAS DE GENERATION INSTRUCTION A (W2)	
472	T2R14	TEST (KA) MAXIMAL (ENTREE DO7)	
	S14M474AS		
474	X	TAILLE TABLE BINAIRE	
	RV674AP14	A (ERO3) DEBORDEMENT TABLE BINAIRE	(4 Δ P3)
	T10R14	MISE EN PLACE CODE INSTRUCTION	
	S14V2	BINAIRE	
	RV482AP14		
	A10V1		
	K10QV1		
	T14R10		
482	T15R4	MISE EN GARDE INDEX	
	A15EM484AS		
484	X	BASE BINAIRE	
	A15R2	ADRESSE INSTRUCTION DANS (RX)	
	V10QV8	MISE EN PLACE INSTRUCTION OU	
	RV493AY14	CONSTANTE DANS TABLE BINAIRE	
	I6QV98304		
	RV5(X)AZ14		
	RV499AP5		
	T9M2+	CHARGEMENT INDEXE (ROP2)	
	RV499Δ		
493	V10QV6		
	A5EV1Y14		
	RV497AY14		
	RV499AP5		
497	A1EV1		
	RV5(X)Δ		
499	T7M1+	CHARGEMENT INDEXE (ROP1)	

500	T6EM0+ A14EM502ΔS	CHARGEMENT INDEXE (RI)	
502	X RV507ΔZ14	INDICATEUR D'ERREUR (IER)	
504	A10V1 A14EVO T14M502Δ	MISE EN PLACE DRAPEAU D'ERREUR (ENTREE EO7) DANS RTR et RAZ IER	
507	T2R8	(KA) → RAD (ENTREE FO7)	
508	I8QV7 T14R15 C8QV3 K10QM1281Δ+ T15R6 T8R15 A15M515ΔS	MISE EN PLACE CODE D'INSTRUCTION (RTR) DANS LA TABLE DES CODES INSTRUCTION DECALAGE INDEXE (TAEDC) RTR	
515	X RV521ΔZ6 A14MO+ S14W16384 A14W32768P14 A14W16384	A (TABCI)	
521	T14MO+ RV528ΔP5 X A5EVO T2R8 A8V1 RV508Δ	MOT DANS TABLE CODES INSTRUCTION VERS SUITE PLACE RESERVEE CAS D'UNE CONSTANTE DOUBLE LONGUEUR	
528	A2R1 T4ER15 RR12	VERS GENERATION DU 2ème CODE INSTRUCTION EVOLUTION (KA)	
531	RV534ΔY5 A1W49152 T1M1+	SORTIE SP FIN INSTRUCTION TEST RSL GENERATION INDICATIONS DANS MOT RESERVE (PROGRAMME SOURCE)	
534	A15V2 T15R14 S14M537ΔS	EVOLUTION (RX) (W2) TEST FIN DU PROGRAMME SOURCE	
537	X RV0ΔN14 RV760Δ	A (B1) . INSTRUCTION SUIVANTE	(4 Δ P1)
540	S4V32768 RV568ΔP4 RCV26Δ V0QV12 RV767ΔY14 T1R14 S14V16 RV767ΔP14 K1QV11 A6R14	A (ER30) TRAITEMENT INSTRUCTIONS DE BRANCHEMENT (N2) TEST OP1 BRANCHEMENT ET RETOUR SP REGISTRE TEST "VIRGULE" A (ER22) TEST MAX1=15 A (ER22) OP1 CADRE EN B12 OP1 → (RI) EN B12	(90 Δ P3) (97 Δ P3) (97 Δ P3)

551	RCV26Δ	BR. ET RETOUR SP. REGISTRE	
	T1R14	TEST MAX2=7	
	S14V8		
	RV779ΔP14	A (ER28)	(109 Δ P3)
	K1QV8	OP2 CADRE EN B9	
	A6R14	OP2 → RI	
	V0QV12	TEST "VIRGULE"	
	RV561ΔZ14		
	S1V2		
	RV781ΔP1	A (ER29)	(111 Δ P3)
	RV461Δ	A (M2)	
561	RCV26Δ	SP REGISTRE	
	T1R14	TEST MAX3=15	
	S14V16		
	RV781ΔP14	A (ER29)	(111 Δ P3)
	K1QV4	OP3 CADRE EN B5	
	A6R14	OP3 → (RI)	
	RV461Δ	A (M2)	
568	A6V10P4	CODE ADRESSE 1	
	S6V4Z4		
	T6EM571ΔS	RANGEMENT (RI)	
571	X	GARAGE RI	
	S6V1	-1 → (RB)	
	A7EV0	O → (RCI)	
	RCV243Δ	BR. ET RETOUR SP EXPRESSION SYMBOLIQUE (BO6)	
	A6EM571Δ	RESTITUTION RI	
	I7QV65535		
	T14R7		
	K7QV11		
	A6R14		
	A12EV551Δ	A (RETOUR)=551 Δ	
	RV26Δ	A (SP REGISTRE)	
	A6V2	INSTRUCTIONS D'ENTREE (O2)	
	A5EV240		
	RV588Δ		
	S4V32768	TEST OPERANDE (P2)	
	RV593ΔP4		
	A5EV128		
588	S7EV1		
	A12EV89Δ	ADRESSE RETOUR=89 Δ (K2)	
	RV26Δ		
	S4V32768	TEST OPERANDE (Q2)	
	RV604ΔN4		
593	A6V8	CODE ADRESSE 1	
	A6W256		
	T6EM596ΔS	RANGEMENT RI	
596	X		
	A7EV0	O → (RCI)	
	RCV243Δ	BR. ET RETOUR SP EXPRESSION SYMBOLIQUE (BO6)	
	A6EM596Δ		

ASSEMBLEUR P2

	K7QV4	REXP CADRE EN B5	
	A6R14		
	A8EV1	(RC)=1	
	RV424Δ	A(L2)	
604	RCV26Δ	BR. ET RETOUR SP REGISTRE	
	T1R14	TEST MAX=7	
	S14V8		
	RV767ΔP14	A(ER22)	(97Δ P3)
	A6V8	CODE ADRESSE 1	
	K1QV4	ROP CADRE EN B5	
	A6R14		
	RV461Δ	A(M2)	
	S4V32768	INSTRUCTION ECRITURE (R2)	
	RV620ΔZ4	VERS TRAITEMENT OPERANDE SYMBOLIQUE	
	A5EV240N4	MAX2=15	
	RV618ΔN4		
	A5EV928	MAX2=58	
	A6V8	CODE ADRESSE 1	
618	A12EV659Δ	A(RETOUR)=659Δ (V2)	
	RV26Δ	A(SP REGISTRE)	
620	T6EM621ΔS	MISE EN GARDE RI	
621	X	GARAGE RI	
	A7EV0	(RCI)=0	
	RCV243Δ	BR. ET RETOUR SP EXPRESSION SYMBOLIQUE	
	A6EM621Δ	RESTITUTION RI	
	A6V4	CODE ADRESSE 1	
	K7QV4	REXP CADRE EN B5	
	A6R14		
	A8EV0	O → (RC)	
	RV424Δ	A(L2)	
	A5EV504	INSTRUCTIONS DE SORTIE (NOMBRES) (S2)	
	RV633Δ		
	A5EV503	(T2)	
633	A7EV0		
	RV637Δ		
	A5EV503	(U2)	
	A7EV1		
637	A6V8		
	RCV26Δ	BR. ET RETOUR SP REGISTRE	
	VQV12	TEST "VIRGULE"	
	RV767ΔY14	A(ER22)	(97Δ P3)
	I5QV15	TEST MAX1=15	
	S14R1		
	RV767ΔN14	A(ER22)	(97Δ P3)
	K1QV21	OP1 CADRE EN B22	
	A6R14		
	A12EV659Δ	A(RETOUR)=659 Δ	
	RV26ΔY7	TEST RCI : BR. SP REGISTRE	
	RCV26Δ	BR. ET RETOUR SP REGISTRE	
	VQV12	TEST "VIRGULE"	

	RV779ΔY14	A (ER28)	(109 Δ P3)
	I5QV496	TEST MAX2	
	T14R9		
	C9QV4		
	S9R1		
	RV779ΔN9	A (ER28)	(109 Δ P3)
	K1QV4	OP2 CADRE EN B5	
	A6R14		
	RCV26Δ	BR. ET RETOUR SP REGISTRE (V2)	
659	I5QV496	TEST MAX2	
	T14R9		
	C9QV4		
	S9R1		
	RV781ΔN9	A (ER29)	(111 Δ P3)
	K1QV11	OP3 CADRE EN B12	
	A6R14		
	RV461Δ	A (M2)	
	S7EV1	(H2)	
	RV3Δ		
	F	RESERVE	
	FE	FIN BANDE PROGRAMME	

VII - TRAITEMENT DES ERREURS

Cette séquence de programme a pour objet la génération du diagnostic d'erreur et sa mise en place dans la table des erreurs en vue de l'impression de la liste de sortie.

Suivant la gravité de l'erreur détectée, les conséquences pour l'assemblage sont différentes.

1 - Classes d'erreurs

Lors de la définition de la table des erreurs (paragraphe IV - 6) nous avons déjà prévu deux classes différentes.

- a) - Une erreur de la classe 2 arrête l'assemblage de l'instruction fausse.
- b) - Une erreur de la classe 1 n'a pas de conséquence sur la traduction de l'instruction puisqu'elle intervient en zone étiquette.
- c) - La classe 0 regroupe les diagnostics correspondant au débordement des diverses tables utilisées par l'assembleur et dont les tailles respectives dépendent du programme à assembler (table du programme source, table des symboles, table binaire, table des erreurs).

Ces débordements ne sont pas à proprement parler des erreurs de programmation mais ils arrêtent aussitôt l'assemblage.

C'est ici que l'on comprend l'un des intérêts de la segmentation des programmes (sans difficulté avec le langage symbolique défini) qui permet de ne pas surcharger la mémoire centrale pendant l'assemblage.

La liste des diverses erreurs et des diagnostics correspondants est fournie dans l'annexe n° 7.

2 - Organigrammes et programmes

Le traitement est évidemment fonction de la classe de l'erreur.

a) - Classe 0

L'assemblage étant stoppé, le diagnostic de l'erreur est immédiatement imprimé sous la forme :

* ERREUR ASSEMBLAGE N *

N caractérise le type d'erreur ($01 \leq N \leq 04$) et n'a pas besoin de figurer dans la table des erreurs.

Pour que le programmeur ait suffisamment de renseignements pour segmenter le programme et que le travail d'assemblage éventuel n'ait pas été fait inutilement, l'apparition du diagnostic est suivie de l'écriture du listing.

La mémoire "système" qui définit le travail à effectuer (voir paragraphe VIII) est remise à zéro tandis que le registre RTR est chargé de la valeur 10 (ce code instruction indique que le programme a été assemblé partiellement. Pour effectuer la mise en place de ce code, la commande est passée au S.P. fin instruction (entrée F07), l'adresse de sortie du sous-programme étant donnée par le registre 12 qui contient l'adresse de début du listing.

Le système défini garde trace d'une erreur de chacune des classes 1 et 2 par instruction.

b) - Classe 1

Le code erreur ayant été mis en place dans le registre d'instruction, il faut passer la zone étiquette qui est cause de l'erreur ; ceci est assuré par la reconnaissance du caractère de séparation "TAB", si ce caractère ne peut être trouvé avant la fin de l'instruction, celle-ci ne peut être assemblée et le programme se débranche au traitement

d'une erreur de classe 2 après avoir ajouté conventionnellement dans le registre RI, le diagnostic d'erreur 20 (ce code a été choisi parce qu'il représente une erreur en zone "code opération" qui est normalement analysée immédiatement après l'étiquette). Dans le cas où l'erreur est bien limitée à la zone étiquette, l'adresse de traitement du code opération est placée dans R12 et le registre RI est transféré en table des erreurs. La mémoire DG indique le nombre de lignes comptant au moins un diagnostic ; l'indicateur d'erreur (IER) est positionné à 1 pour rappeler qu'il y a déjà une erreur de classe 1 dans l'instruction en cours d'assemblage. Le contenu du registre 12 donne l'adresse de branchement à la suite du traitement (dans le cas de l'erreur 10, le contenu de ce registre n'est pas l'adresse de traitement des codes opérations mais l'adresse de retour dans la séquence correspondant à la directive "FIN").

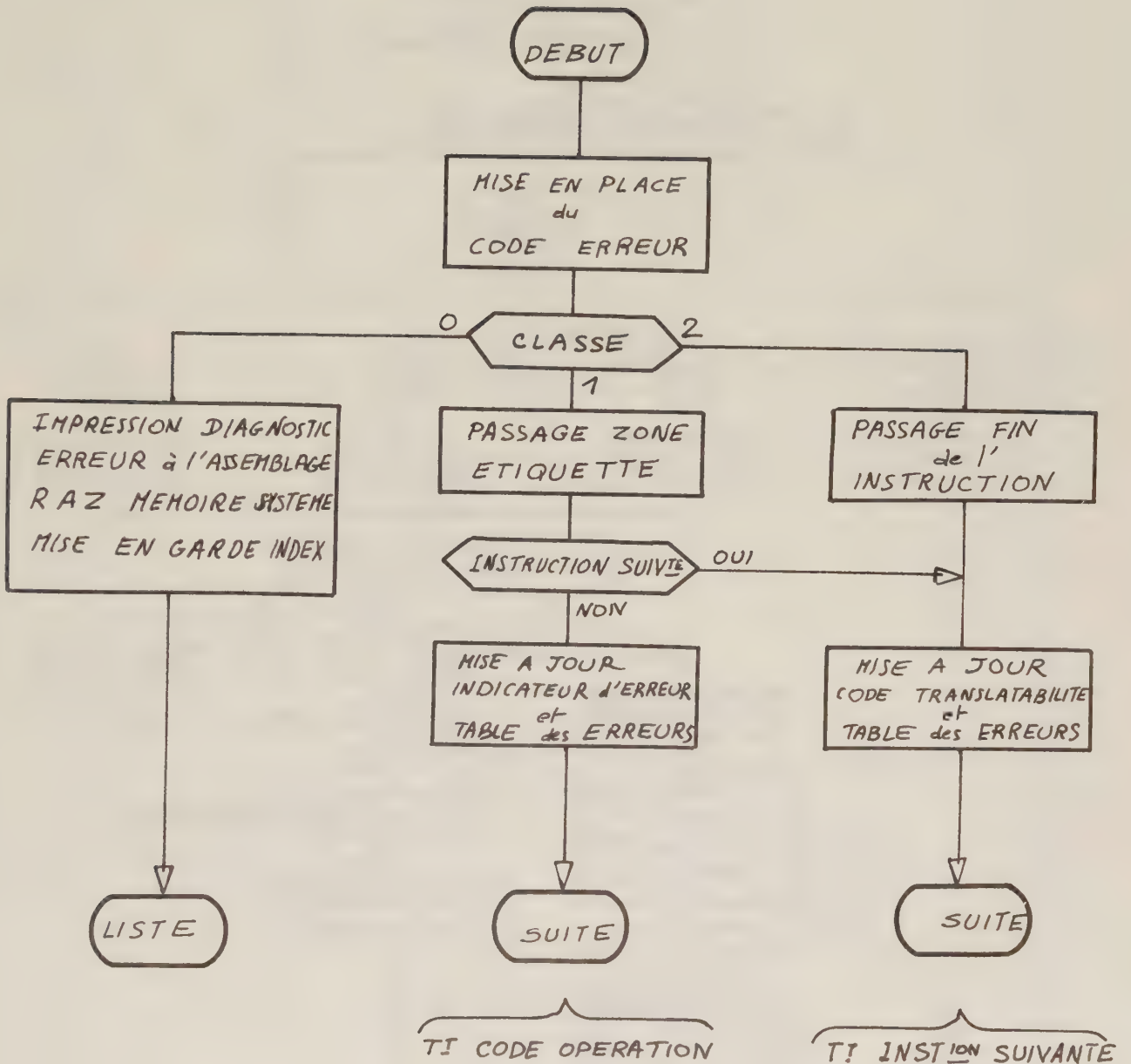
c) - Classe 2

Après génération du code erreur la fin de l'instruction est ignorée. Comme pour la classe 1, la valeur de DG est testée, indiquant ainsi si le nombre maximal de lignes d'erreurs n'est pas atteint (sinon la commande est passée à la séquence "erreur 04").

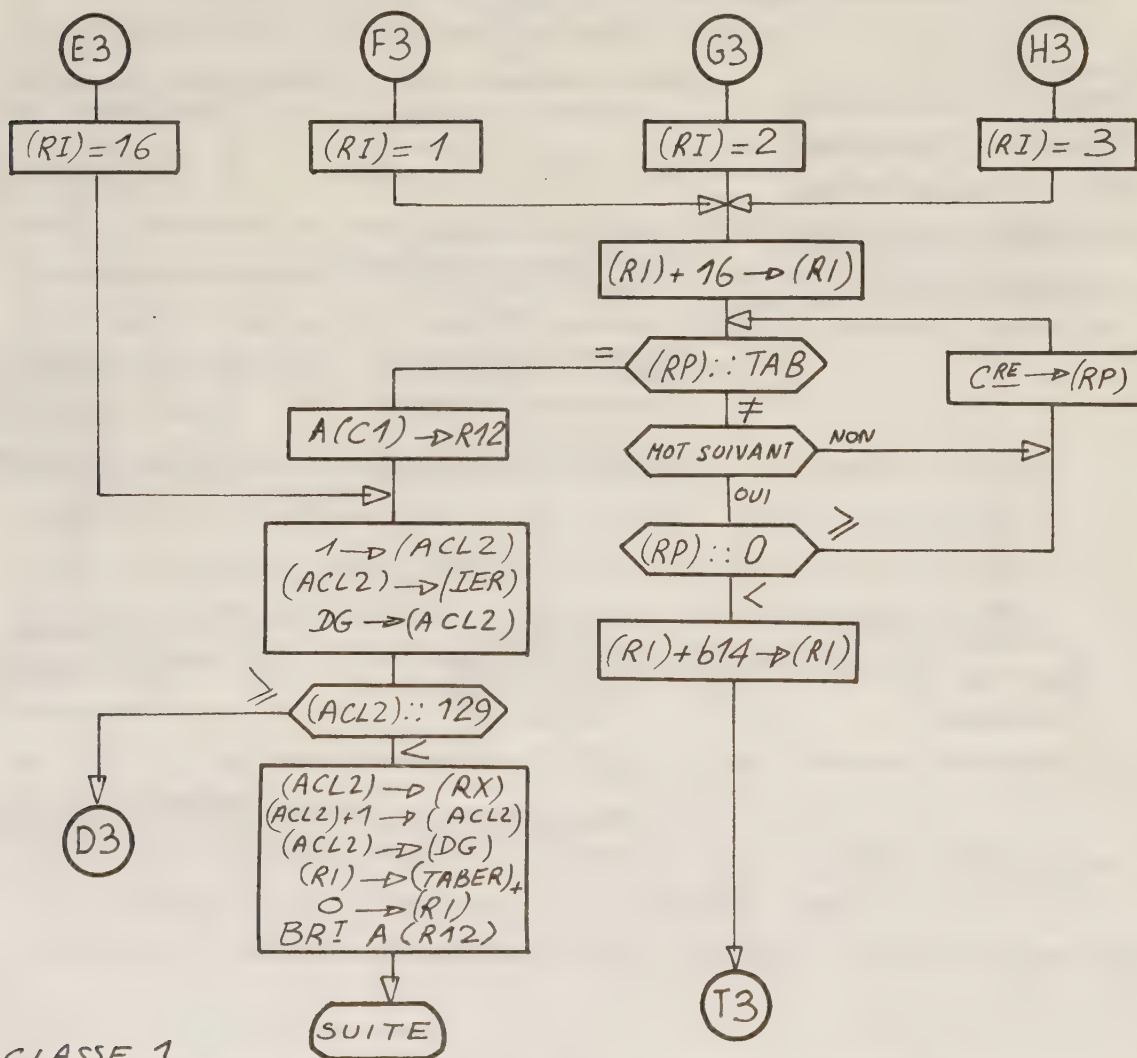
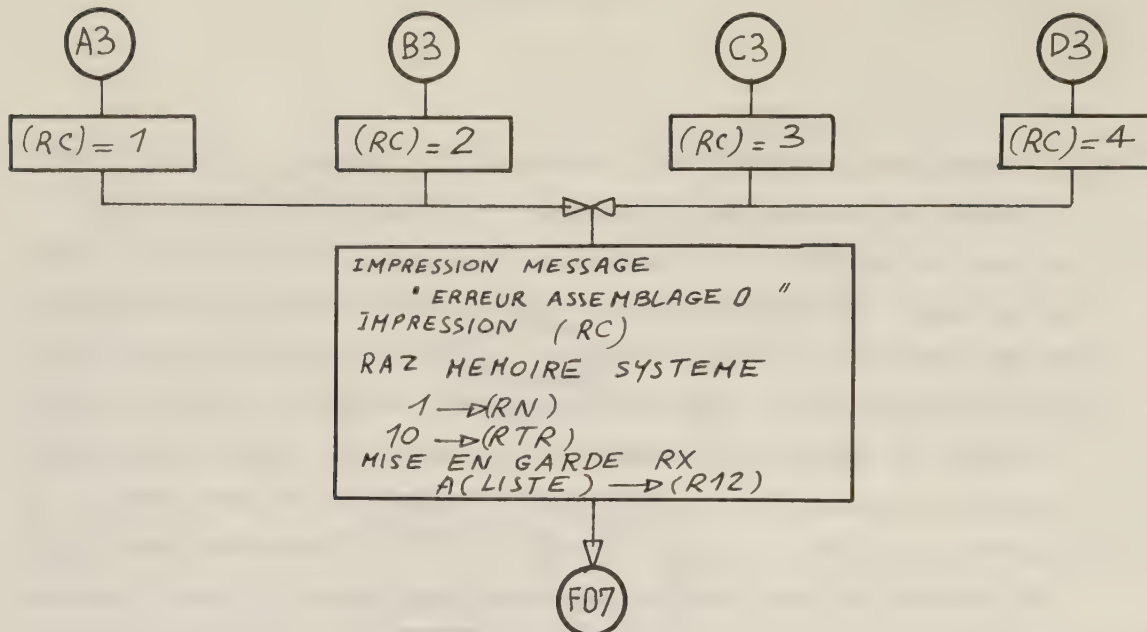
Si (IER) = 0 (pas d'erreur de classe 1 dans l'instruction) DG est augmenté de 1 et RI est placé dans la table. Dans le cas où une erreur de classe 1 a été détectée (IER = 1), DG n'évolue pas et le code erreur est ajouté dans la table à celui déjà existant et le sous-programme FIN D'INSTRUCTION (entrée EO7) est appelé.

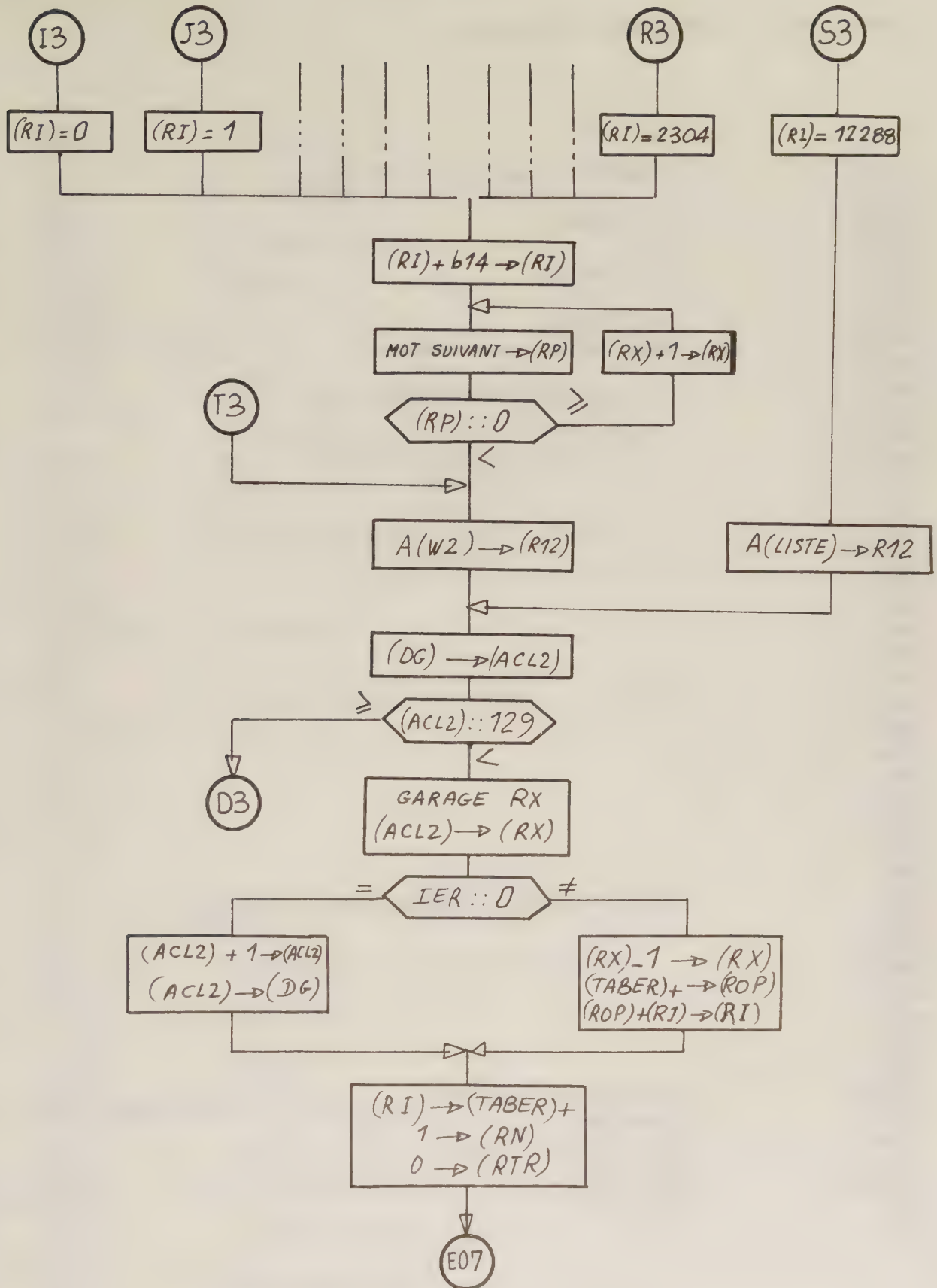
La sortie de ce sous-programme est désignée par l'adresse contenue dans le registre 12 :

- début listing s'il s'agit de l'erreur 30 (pas de pseudo-instruction "FIN" comme dernière instruction du programme source).
- évolution index table programme source (W2) dans tous les autres cas.



13 _ ASSEMBLEUR P3 _ ORG. Simplifié





ERREURS de la CLASSE 2-

15 - ASSEMBLEUR P3 - ORG. Machine (2)

ASSEMBLEUR P3 : TRAITEMENT DES ERREURS

0	A8EV1	ERREURS CLASSE 0 - A3 (ERO1)	
	RV7Δ		
2	A8EV2	B3 (ERO2)	
	RV7Δ		
4	A8EV3	C3 (ERO3)	
	RV7Δ		
6	A8EV4	D3 (ERO4)	
	EV23	IMPRESSION DU MESSAGE :	
	EV49		
	EV24	* ERREUR ASSEMBLAGE <u>ON</u> *	
	EV15		
	EV37		
	EV50		
	EV50		
	EV37		
	EV53		
	EV50		
	EV15		
	EV33		
	EV51		
	EV51		
	EV37		
	EV45		
	EV34		
	EV44		
	EV33		
	EV39		
	EV37		
	EV15		
	EV0		
	ER8	<u>N</u> =(R8)=NUMERO ERREUR DANS LA CLASSE 0	
	EV15		
	EV23		
	EV49		
	EV24		
	EV25		
	EV25		
	A8EV0		
	T8EM0Δ	A (MEMOIRE SYSTEME)	CHARGEUR (55 Δ P1)
	A1EV1	1 → (RN)	
	T15R4	MISE EN GARDE RX	
	A12EV0Δ	A (LISTE)	(0 Δ P4)
	A10EV10	(RTR)=10	(507 Δ P2)
	RV0Δ	A (FO7)	
	F		
	F		
	F		
	F	RESERVES	
48	A6EV16	ERREURS CLASSE 1 - E3 (ER10)	
	RV59Δ		

50	A6EV1 RV55Δ	F3 (ERREUR 11)	
52	A6EV2 RV55Δ	G3 (ERREUR 12)	
54	A6EV3 A6V16	I3 (ERREUR 13)	
56	VOQV30 RV73ΔY14	TEST TAB	
	A12EV0Δ	A(C1)	(92 Δ P1)
59	A14EV1 T14M0Δ	A(1ER)	(502 Δ P2)
	A14EM62ΔS		
62	X T14R10 S10V129 RV4ΔP10 T15R10 T14R15 A14V1 T14M62Δ T6EM136Δ+ T10ER15	DG (NOMBRE DE LIGNES COMPORTANT UN OU PLUSIEURS DIAGNOSTICS) TEST NOMBRE D'ERREURS MISE EN PLACE DIAGNOSTIC DANS LA TABLE DES ERREURS A(TABER)	
73	RR12 V3QV5 T13R0Y14 C0TV25Y14 A3V1Y14 RV56ΔY14 A0EM1+ RV84ΔN0 A15V1 C0TV24 A3EV1 RV56Δ	BT. A L'ADRESSE CONTENUE DANS R12 PASSAGE ZONE ETIQUETTE	
84	A6V8192 RV116Δ	PAS DE RECONNAISSANCE DE ZONE	
86	S12V0Δ A12V805 RV50ΔN12 RV93Δ	A(ER11-20) A(ER11) A(ER20)	
90	A6EV12288 A12EV0Δ RV117Δ	S3 (ERREUR 30) A(T3)	
93	A6EV0 RV112Δ	I3 (ERREUR 20)	
95	A6EV256 RV112Δ	J3 (ERREUR 21)	
97	A6EV512 RV112Δ	K3 (ERREUR 22)	
99	A6EV768	L3 (ERREUR 23)	

101	RV112Δ A6EV1024	M3 (ERREUR 24)	
103	RV112Δ A6EV1280	N3 (ERREUR 25)	
105	RV112Δ A6EV1536	O3 (ERREUR 26)	
107	RV112Δ A6EV1792	P3 (ERREUR 27)	
109	RV112Δ A6EV2048	Q3 (ERREUR 28)	
111	RV112Δ A6EV2304	R3 (ERREUR 29)	
113	A6V8192 A0EM1+ A15V1P0 RV113ΔP0	PASSAGE FIN D'INSTRUCTION	
116	A12EV0Δ	A(W2) - T3	(534 Δ P2)
117	A14EM62Δ T14R7 S7V129 RV6ΔP7 T15R4 T14R15 A0EM0Δ RV128ΔY0 A14V1 T14M62Δ RV131Δ	TEST NOMBRE DE DIAGNOSTIC	
128	S15V1 A1EM136Δ+ A6R1		
131	T6EM136Δ+ A1EV1 A5EV0 A10EV0 RV0Δ FE	CODE(S) ERREUR(S) EN TABLE DES ERREURS	
		A(E07) FIN BANDE PROGRAMME	(504 Δ P2)

VIII - ASSEMBLAGE 2ème PHASE - EDITION DES LISTES

Avant d'envisager les questions de "listing" proprement dites, il est nécessaire de définir les codes qui caractérisent la nature des instructions rencontrées lors du chargement car ils jouent un rôle important dans l'édition de la liste d'assemblage.

1 - Codes instruction (Liste en annexe n° 8)

Ils fournissent les indications concernant :

- a) - translatabilité : la zone d'adresse d'une instruction peut être
 - non translatable (absolue)
 - translatable
 - adresse de chaînage
- b) - les constantes :

Une constante définie en simple longueur se comporte du point de vue chargement comme une instruction non translatable, par contre sur la liste de sortie elle en diffère par l'absence de signification particulière du champ qui définit le code opération d'une instruction exécutable.

Certaines directives génèrent plusieurs mots pour une seule ligne de programme. Afin de gagner du temps à l'assemblage, une constante avec facteur itératif n'est placée qu'une fois dans la table binaire et ce n'est qu'au chargement qu'elle est répétée autant de fois qu'il est nécessaire.

La réservation de mémoire et la définition de constantes sont indiquées par un même code instruction ; le contenu correspondant de la table binaire, fixe le facteur et éventuellement la constante (figure 12).

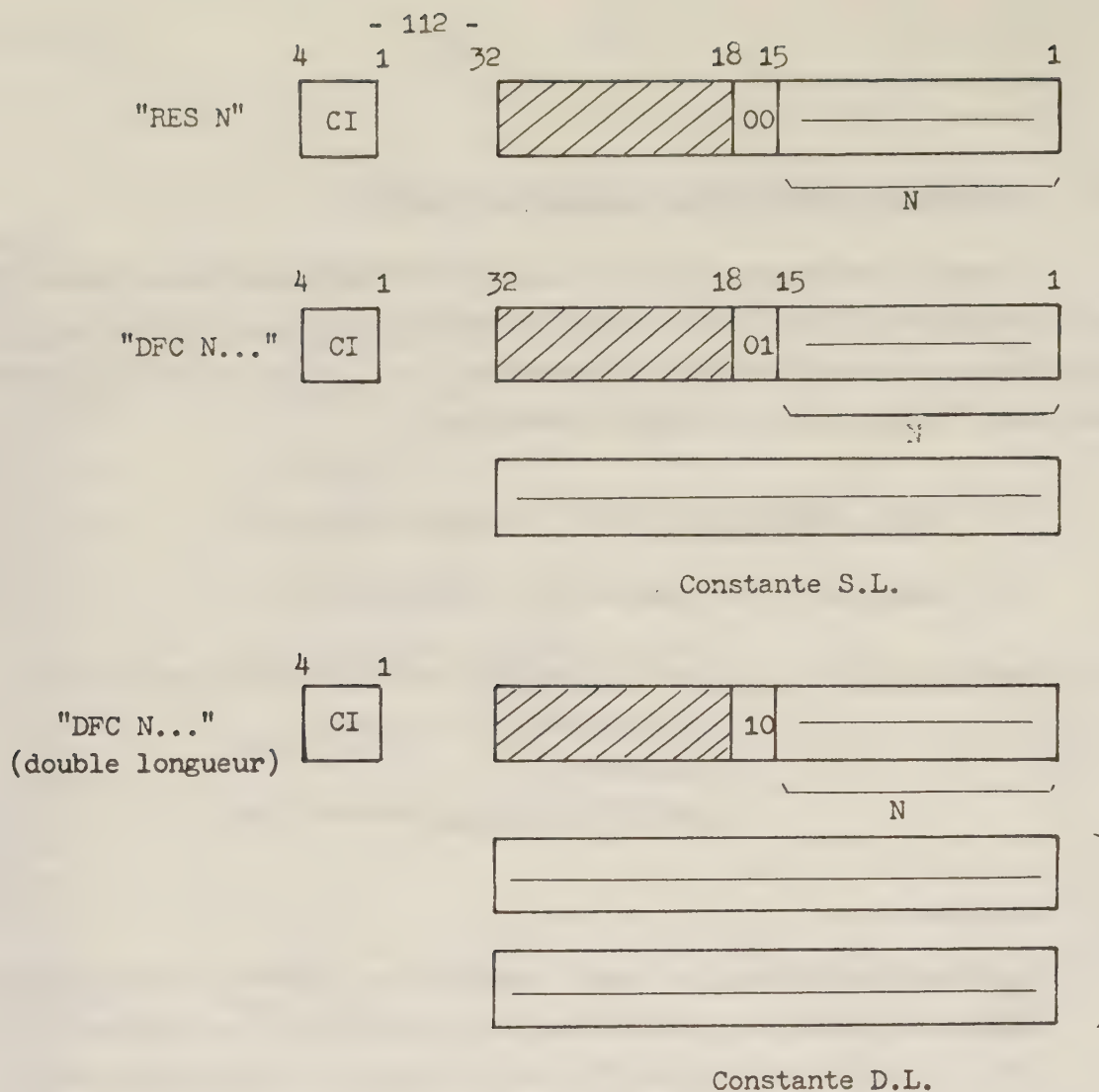


Figure 12

Dans la table binaire, il n'est pas possible de faire la différence entre, une suite de constantes quelconques, une (ou plusieurs) constante en double longueur et une chaîne de caractères générée par une directive TEXTE.

Cette distinction n'a aucun intérêt au niveau du chargement mais elle est utile pour l'écriture de la liste d'assemblage.

Pour que le contenu des mots générés soit bien imprimé avec l'instruction source correspondante, le nombre de constantes créées est placé dans les mots de séparation des lignes du programme source (voir "lecture du programme source" ch. III paragraphe III-2) comme le montre la figure 13.

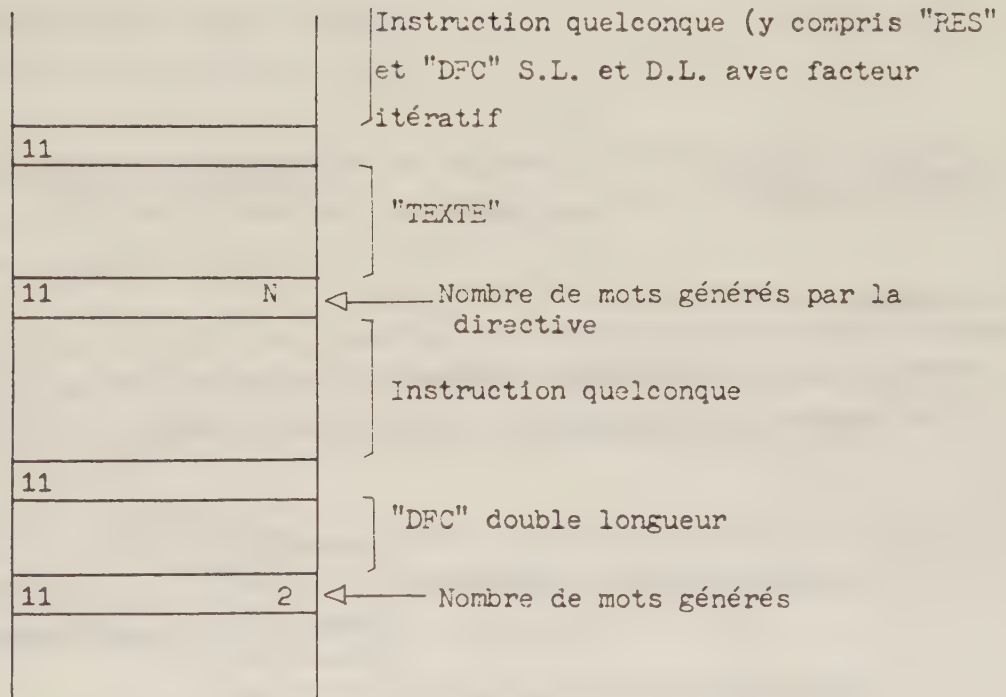


Table programme-source

Figure 13

Ainsi l'impression de la liste d'assemblage correspondant à une directive sera réalisée de façon à faire apparaître l'instruction source en regard de la première constante (figure 14)

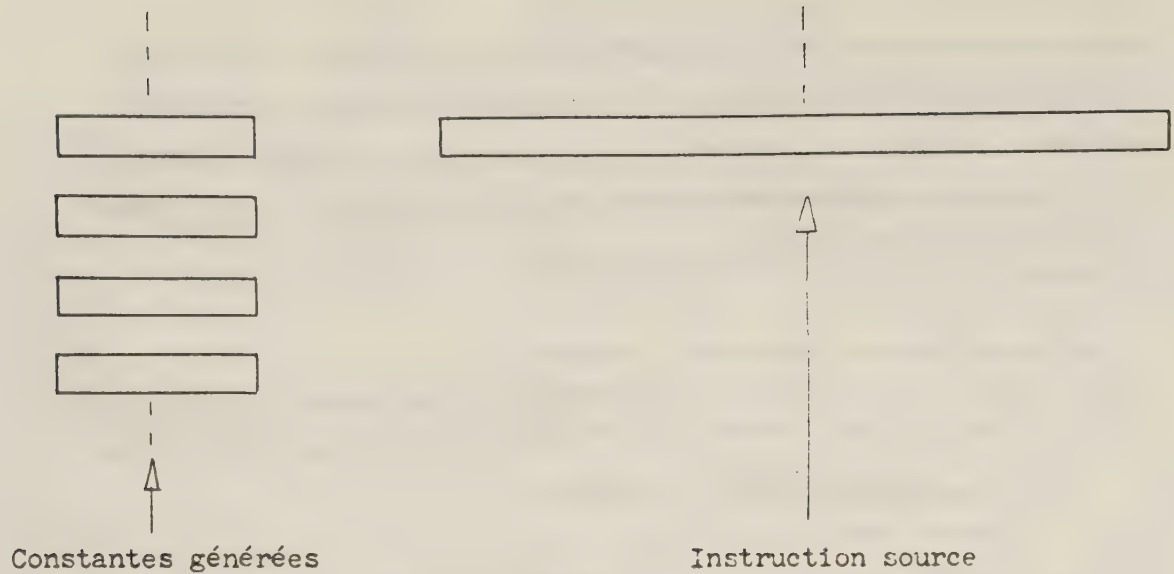
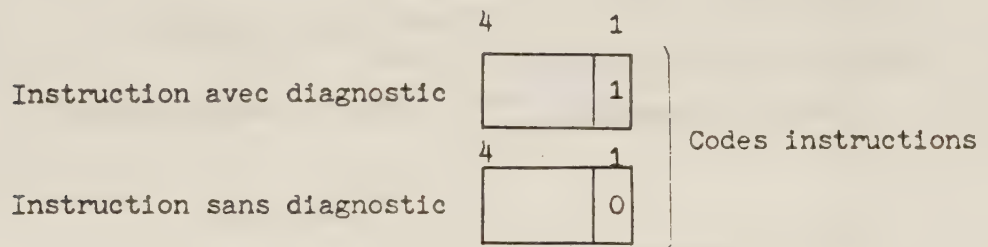


Figure 14

c) - erreurs :

dans le cas où l'instruction comporte au moins une erreur, l'existence d'un diagnostic est indiquée par une marque en début du code instruction.



d) - les autres éventualités à signaler concernent directement le chargement : début et fin des instructions du programme objet, définition et références externes.

2 - Eclatement et classement des tables de symboles

Nous avons déjà étudié la façon dont est constituée la table des symboles qui regroupe :

- symboles définis
- définitions externes
- références externes
- symboles non définis.

Tous les symboles non définis à la fin de la première phase de l'assemblage sont considérés comme externes au programme.

a) - Eclatement de la table

La table des symboles donne naissance à trois tables :

- table des symboles
- table des définitions externes
- table des références externes.

Les deux dernières fournissent les adresses qui permettent éventuellement au chargeur d'assurer les liaisons entre les segments de programme assemblés séparément.

La taille de la table des symboles est fournie par sa première adresse (connue) et par la position du pointeur PTRS. Les adresses de fin des trois tables sont tenues à jour par les pointeurs PTR2 - PTR3 - PTR4.

Le pointeur PTR2 contient au départ la première adresse de la table des symboles, si la table ne comporte pas d'élément (PTR2 = PTRS) on passe directement à l'édition des listes.

Le registre RCT (registre de contraction table) préalablement remis à zéro fournit le nombre de symboles extraits de la table.

La méthode de séparation consiste à analyser la zone d'identification des mémoires contenant les adresses des symboles :

Si B19 = 1, il s'agit d'une définition externe, le symbole et son adresse sont placés dans la table correspondante ; il faut ensuite faire évoluer PTR3, RCT et PTR2.

Si B19 ≠ 1, le test du digit B18 indique si le symbole est "non défini" dans ce cas il est traité comme une référence externe (B20 = 1) ; après transfert en table, PTR4, RCT et PTR2 sont incrémentés.

Si le symbole n'appartient à aucune de ces catégories, il reste dans la table primitive ; toutefois si $(RCT) \neq 0$ le symbole est déplacé puisque des cellules ont été libérés.

- b) - Quand le travail d'éclatement est terminé, chacune des tables est classée par ordre alphabétique au moyen d'un sous-programme:

Le nombre d'éléments de la table est préalablement calculé et chargé dans le registre RN. S'il n'y a qu'un symbole la sortie du sous-programme est immédiate.

Dans le cas général, le classement est effectué par tri alphanumérique. Les symboles étant cadrés à gauche dans le mot, la comparaison entre les équivalents numériques de la chaîne des caractères représentant le symbole permet de réaliser le classement.

La méthode consiste à comparer deux éléments consécutifs (i et $i + 1$), si le premier est numériquement supérieur ou égal au second les cellules sont permutées sinon on passe à la comparaison suivante ($i + 1$, $i + 2$). A la fin d'une exploration de la table, l'élément supérieurement extrême est certainement à sa place et ainsi de suite ; le classement est donc terminé, soit après $(N - 1)$ cycles d'exploration (cas le plus défavorable), soit à la fin d'un cycle sans permutation. Si le contenu du registre RCP (permutation) est nul à la fin d'une exploration, la table est correctement classée et il est inutile de continuer le traitement.

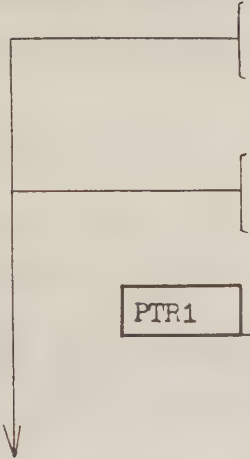
Les possibilités de multidéfinition (digit B21), ne sont pas testées à ce niveau, car elles n'apportent aucun renseignement utile au traitement.

A titre d'exemple, nous représentons dans la figure 15 les tables générées par l'éclatement d'une table; en fin de traitement, le registre de contraction contient le nombre de cellules extraites soit 4.

TABSB

T	E	S	T	1
			0	0
A	L	P	H	A
			0	0
B	E	T	A	—
			0	0
T	A	B	E	X
	4		0	0
R	A	D	—	—
			0	0
J	O	K	—	—
	2		0	0
J	U	M	P	1
			0	0
G	A	M	M	A
	2		0	0
A	R	C	—	—
	4		0	0

PTR1



G	A	M	M	A
	2		0	0
J	O	K	—	—
	2		0	0

DEFINITIONS EXTERNES

TBDFX

PTR3

TBRFX

PTR4

A	R	C	—	—
			0	0
T	A	B	E	X
	4		0	0

REFERENCES EXTERNES

TABSB

A	L	P	H	A
			0	0
B	E	T	A	—
			0	0
J	U	M	P	1
			0	0
R	A	D	—	—
			0	0
T	E	S	T	1
			0	0

PTR2

SYMOLES

Figure 15

L'assemblage normal d'un programme symbolique fournit deux listes :

- La liste d'assemblage imprimée qui comporte les instructions binaires générées, en regard du programme source et éventuellement la table des symboles.
- La liste objet binaire perforée sur ruban qui est directement utilisable par le chargeur. Elle est constituée des instructions binaires essentiellement.

3 - Liste d'assemblage

a) - Composition

- Le titre indique le nom symbolique du programme (s'il existe), le type de la machine (CAB 502B mise en service en Octobre 1963) qui a fait l'assemblage ainsi que la date et le numéro de page:

PROGRAMME - NOMPR - ASSEMBLE PAR CAB 502B OCT 63

- La mise en page est la suivante :

EMPL .	CODE	OBJET	ADRI	NQPS.	PROGRAMME SOURCE
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____
_____	_____	_____	_____	_____	_____
↑	↑	↑	↑	↑	↑
compteur d'adresses	code opération (INSTRUCTIONS)	instruction complète	adresse mémoire	N° de ligne	Programme source

SYMBOLES

_____	_____
_____	_____
_____	_____
↑	↑
	ADRESSE
	NOM SYMBOLIQUE

DEFINITIONS EXTERNES

_____	_____
_____	_____
_____	_____
_____	_____

REFERENCES EXTERNES

_____	_____
_____	_____
_____	_____

DIAGNOSTIC N

- Toutes les grandeurs binaires sont représentées en code hexadécimal.

b) - Mode opératoire

Après remise à zéro de KA (compteur d'adresses), KN (compteur de lignes du programme source) et KER (compteur de lignes erronées) le traitement commence par la détection des commentaires. Si le 1er caractère de la ligne de programme définit la présence d'une ligne de commentaires, le programme passe directement à l'impression de celle-ci. Dans le cas contraire, l'instruction binaire est placée dans le registre d'instruction (RI) et le code correspondant est extrait dans RT (registre de test) ; c'est la valeur de ce code qui oriente le traitement vers les erreurs (M4), les codes spéciaux (M4), les constantes multiples (O4) et les instructions quelconques.

- Instructions et constantes simples

Cette séquence réalise l'impression consécutive du contenu du compteur d'adresse, du code opération s'il s'agit d'une instruction (d32 à d26 pour les instructions quelconques, d32 à d27 pour les branchements), du mot binaire généré, de la partie adresse pour une instruction faisant intervenir un opérande en mémoire.

Après évolution de KA d'une unité, la séquence assure l'impression de l'instruction du programme source (H4) et termine sur le test de fin de liste (l'édition est terminée si tout le programme source a été imprimé).

- Erreurs

Si le code instruction comporte le drapeau d'erreur (digit B1), la cellule correspondant à la valeur du compteur d'erreurs (KER) est extraite de la table des erreurs. S'il n'y a pas d'erreur de type 2, le diagnostic "ER1N" est imprimé avant la valeur du compteur KA et la commande est passée au traitement d'une instruction normale (F4) ou d'une constante multiple (O4). Si la ligne comporte une erreur de type 2, la valeur actuelle de l'adresse n'est pas imprimée et le diagnostic "ERREUR 2N" prend la place du code objet non généré (les diagnostics apparaissent ainsi très clairement dans la liste d'assemblage) et le programme s'oriente sur l'impression du programme source (G4).

- Codes spéciaux

A l'assemblage les codes de gestion sont de deux types. Le code "fin de programme" (12₁₀) indique que le mot suivant contient l'adresse d'exécution du programme (si celui-ci est débordé) qui est éventuellement imprimée à la place du contenu du compteur d'adresse. Le code "programme non assemblé" (10₁₀) précise qu'une erreur a stoppé l'assemblage et qu'à partir de la rencontre de ce code, il ne faut imprimer que le programme source. (C'est la mise à 1 de la mémoire FBIN qui traduit cette éventualité).

- Constantes multiples

Après impression de (KA) le test des digits B16 et B17 du registre d'instruction permet de différencier l'évolution simple du compteur d'adresses (directive "RES") d'une génération de constante avec facteur itératif. Dans le premier cas le compteur KA est modifié par la quantité placée en zone adresse de RI ; dans le deuxième cas le mot binaire suivant est imprimé et il est suivi de l'instruction du programme source pour une constante en simple longueur et éventuellement de la suite de la constante en double longueur à la ligne suivante. De toute façon, il y a évolution de (KA) et saut d'une ligne pour rappeler la présence du facteur itératif.

Quand l'ensemble des programmes objet et source a été édité, la commande est passée à l'impression des tables. Celle-ci est exécutée par appel à un sous-programme. Après test de l'existence de la table (au moins un élément), il assure l'impression du titre (SYMBOLES, REFERENCES EXTERNES, DEFINITIONS EXTERNES) et de la succession des couples "symbole alphanumérique-adresse". Si le symbole est multiplement défini ($B21 = 1$) le diagnostic "*MULTIDEF.*" est placé en regard du symbole (l'adresse imprimée est celle de la première définition du symbole).

4 - Bande binaire

Après impression du nombre de diagnostics émis au cours de l'assemblage, le système peut générer une bande à partir du programme objet binaire si celui-ci ne comporte aucune erreur. Après ce test et après vérification que la liste binaire a été demandée (contenu de la "mémoire système"), le message : ORGANE DE PERFORATION ? est édité. Le choix entre impression et perforation ne pouvant être assuré par programmation, il faut que l'utilisateur connecte manuellement un organe de perforation ; après cela, il frappe sur le clavier de la machine à écrire "OK", la lecture de ce message conventionnel constitue le point de départ de la génération de la bande.

a) - Outre les "avance-bande" de début et de fin de bande nécessaires aux manipulations ultérieures, nous trouvons :

- le code début de bande à charger (B16), le nombre d'ordres du programme (4 chiffres hexadécimaux) et le nom symbolique du programme (ou des codes espaces) sur 8 caractères hexadécimaux.
- la suite des instructions, chacune étant précédée du code qui la caractérise.
- le code "définitions externes" (E16), la taille de la table correspondante (2 chiffres) et la suite des couples : symbole-adresse (8 chiffres pour le symbole, 4 pour l'adresse).

- le code "références externes" (F16), la taille de la table et la suite des couples "symbole-adresse".

Le code "fin de chargement" (D16) n'est pas présent en fin de bande parce que, a priori, plusieurs bandes peuvent être utilisées pour le chargement d'un seul programme en mémoire. Ce code sera donc transmis à la machine par l'utilisateur (frappe au clavier de la flexowriter, par exemple) quand il jugera que tous les éléments ont été fournis et qu'il n'y a plus d'autre section de programme à charger.

- b) - Le code B, le nombre d'instructions et le nom symbolique ayant été perforés, le traitement continue par l'extraction du code caractère et du mot machine correspondant; après écriture, le code instruction est testé.

Pour un code instruction quelconque, il y a incrémentation du compteur d'adresse et passage à l'instruction suivante. Si le code est celui du chargement de constantes (8) des tests similaires à ceux utilisés lors de l'édition de la liste d'assemblage permettent de perforer éventuellement la constante (sur 8 ou 16 caractères) et d'assurer l'évolution du compteur d'adresse.

Lorsque le système rencontre le code "fin de programme", il se débranche vers l'impression des tables binaires (L4).

Le sous-programme qui traite les tables binaires consiste à imprimer le code correspondant à la table générée, la taille de la table (pointeur et première adresse relatifs à la table) et à sortir successivement chaque cellule à raison de 8 caractères hexadécimaux pour le symbole (32 digits) et 4 caractères pour l'adresse (adresse maximum $2^{15}-1$).

5 - Organigrammes et programmes

Le traitement ayant été expliqué précédemment, voyons quelques points particuliers de la programmation.

Pour une instruction exécutable l'impression du code objet par zones déterminées est difficile puisque les limites de ces zones varient avec les instructions et ne correspondent pas aux limites des caractères hexadécimaux que contient le mot machine.

Nous avons donc choisi d'imprimer à part, le code opération et l'adresse du premier opérande si celui-ci est symbolique.

a) - Test du code opération

Lors de l'étude du langage d'assemblage (chapitre II), nous avons remarqué que la zone code opération occupe 7 digits (B32 à B26) pour les instructions quelconques et 6 digits (B32 à B27) pour les branchements.

Pour n'imprimer que les codes correspondant à la zone correcte, il faut tester le code opération pour savoir s'il s'agit d'un branchement ou non.

Les codes opération des deux instructions de branchement existantes sont :

$$OP1 = 010010$$

$$OP2 = 110010$$

Il y a branchement s'il existe dans le mot-machine l'une des combinaisons suivantes : (A, B, C, D, E, F, G représentant les digits B32, 31, 30, 29, 28, 27, 26)

$$\begin{aligned}(B^t) &= \bar{A} B \bar{C} \bar{D} E \bar{F} G \vee \bar{A} B \bar{C} \bar{D} E \bar{F} \bar{G} \vee A B \bar{C} \bar{D} E \bar{F} G \vee A B \bar{C} \bar{D} E \bar{F} \bar{G} \\ &= \bar{A} B \bar{C} \bar{D} E \bar{F} (G \vee \bar{G}) \vee A B \bar{C} \bar{D} E \bar{F} (G \vee \bar{G}) \\ &= B \bar{C} \bar{D} E \bar{F} (\bar{A} \vee A)\end{aligned}$$

$$\underline{(B^t) = B \bar{C} \bar{D} E \bar{F}}$$

Le test peut donc être organisé de la façon suivante :

Il faut donc réaliser un test d'égalité à 10 et un test d'égalité à 4 après intersection du registre RI avec le masque : 1100

Les instructions de lecture en virgule normalisée simple et double longueur représentent un cas particulier : le code d'adresse 1 est toujours 1000, et c'est le digit B26 qui indique si l'opérande est en registre ou en mémoire (B26 = 1 pour un opérande en mémoire).

Pour ces deux instructions, le code opération est différent suivant le type d'opérande ; il suffit donc d'ajouter aux tests précédents, le test particulier à ces opérations (codes : 0110011 pour lecture simple longueur et 1010011 pour lecture double longueur _opérande en mémoire).

c) - Saut de page

L'impression n'est possible que sur la machine à écrire, il n'y a donc pas de saut de page automatique.

Le sous-programme SP "RC" permet de réaliser le saut de page ; il compte le nombre de lignes imprimées pour chaque page et le nombre de pages du listing.

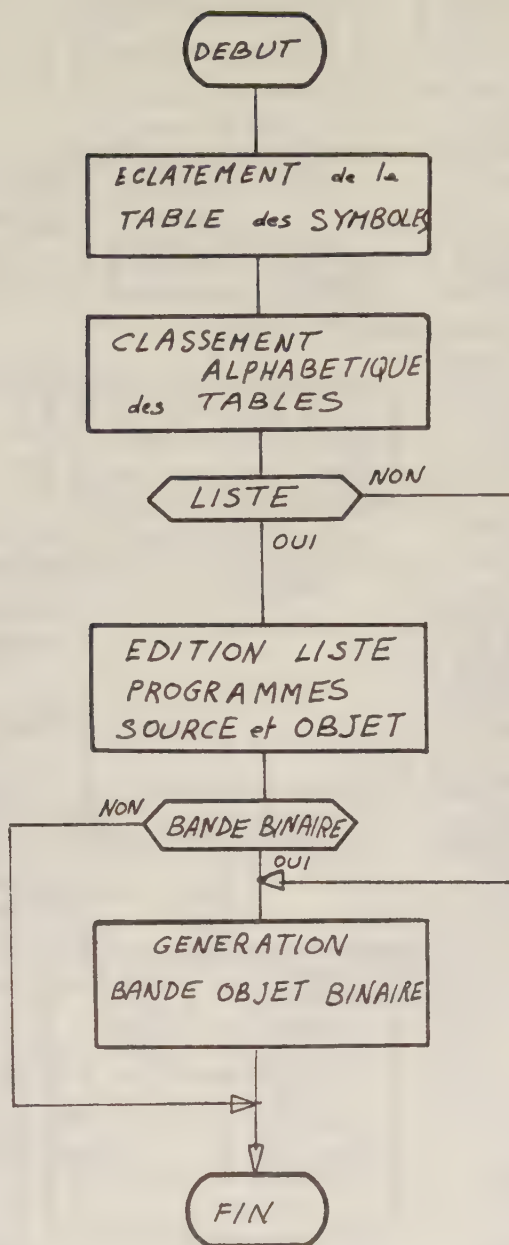
Le saut de page s'obtient par la succession de retours de chariot ; il sera effectivement correct si ce papier utilisé est du format 51-31 et s'il a été placé convenablement au début de l'édition. L'utilisation d'un autre format nécessiterait la modification de quelques instructions du sous-programme puisque le nombre de retours de chariot à effectuer en fin de page et en début de la suivante, dépend du nombre de lignes contenues dans le format.

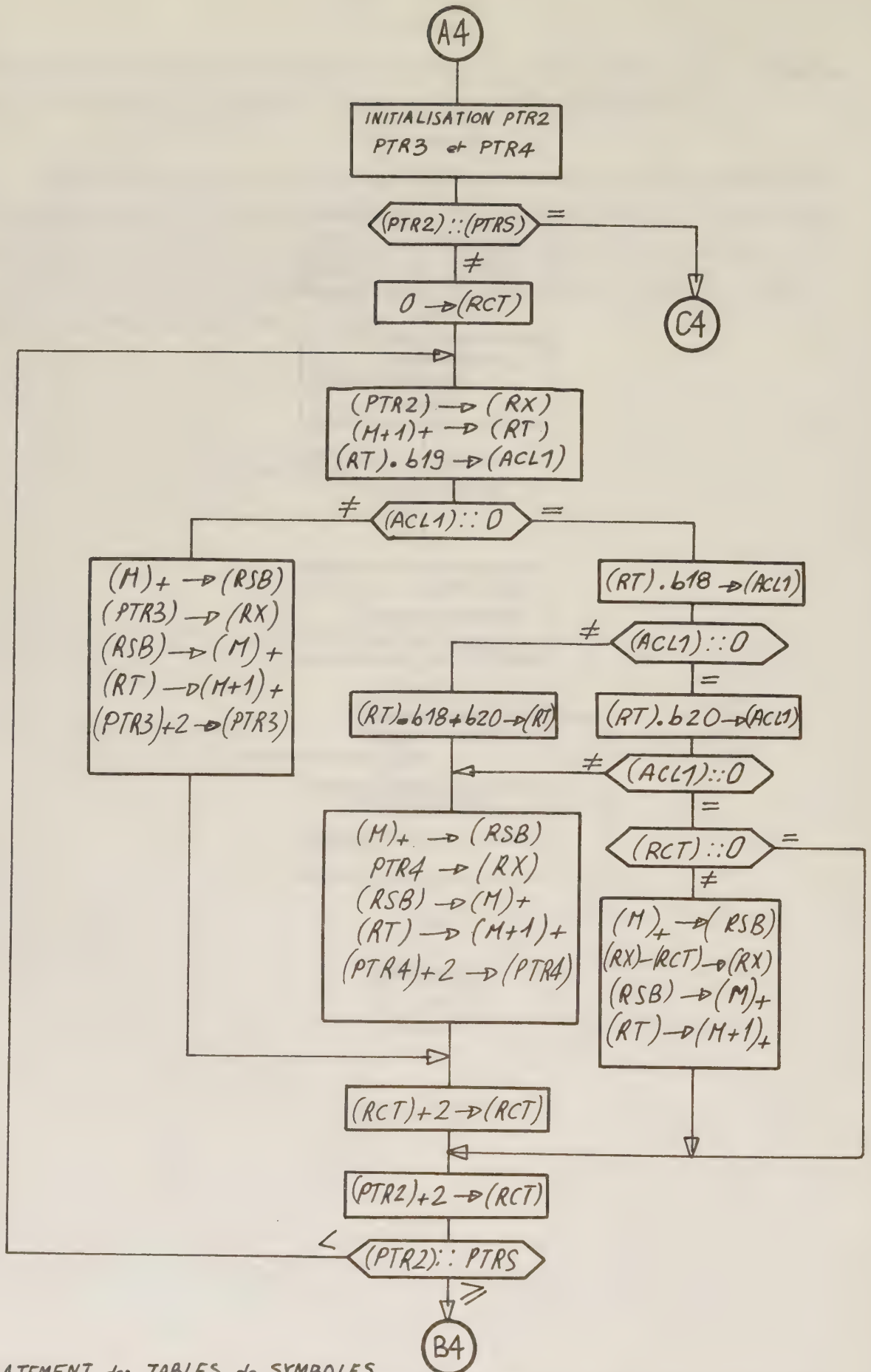
d) - Simplification dans le cas des modèles B1

Le choix entre organe de perforation et machine à écrire, peut être programmé sur les CAB500 modèle B1. Il devient alors inutile d'avoir

deux sections de programme différentes pour la sortie des listes, l'impression et la perforation peuvent se faire en quasi-simultanéité.

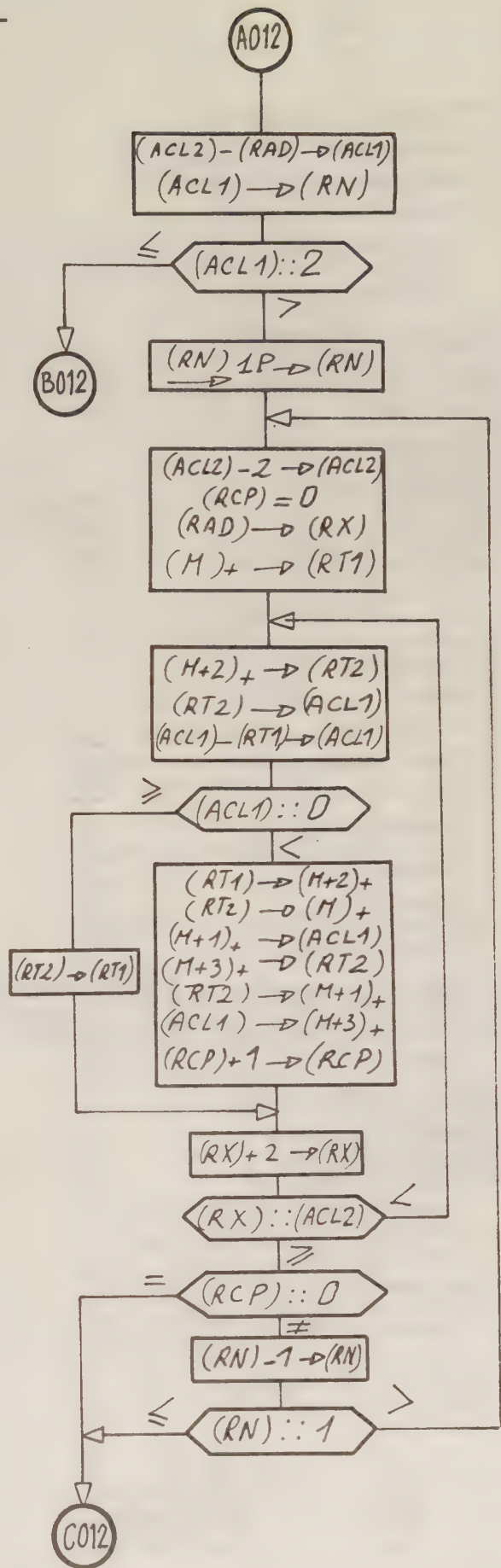
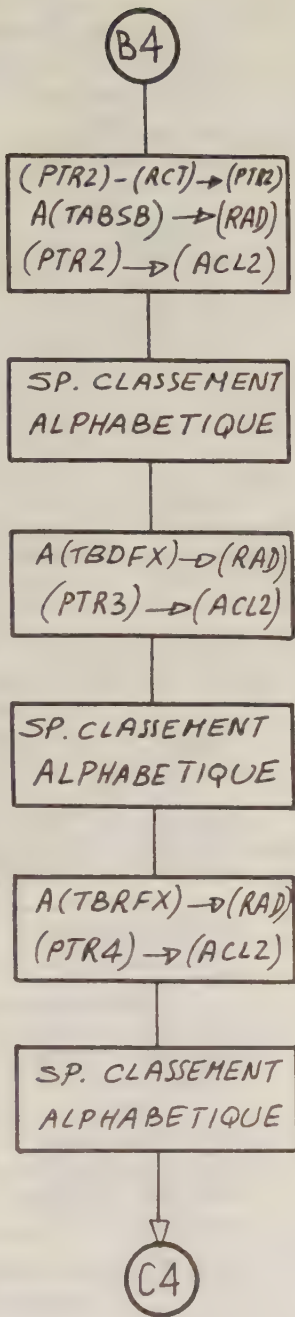
Les instructions binaires sont perforées dans la même séquence que l'impression sur la liste d'assemblage ; il en est de même des tables des définitions et références externes. Le programme est alors raccourci, et le gain de temps est appréciable puisque les différentes tables ne sont appelées qu'une fois au lieu de deux.



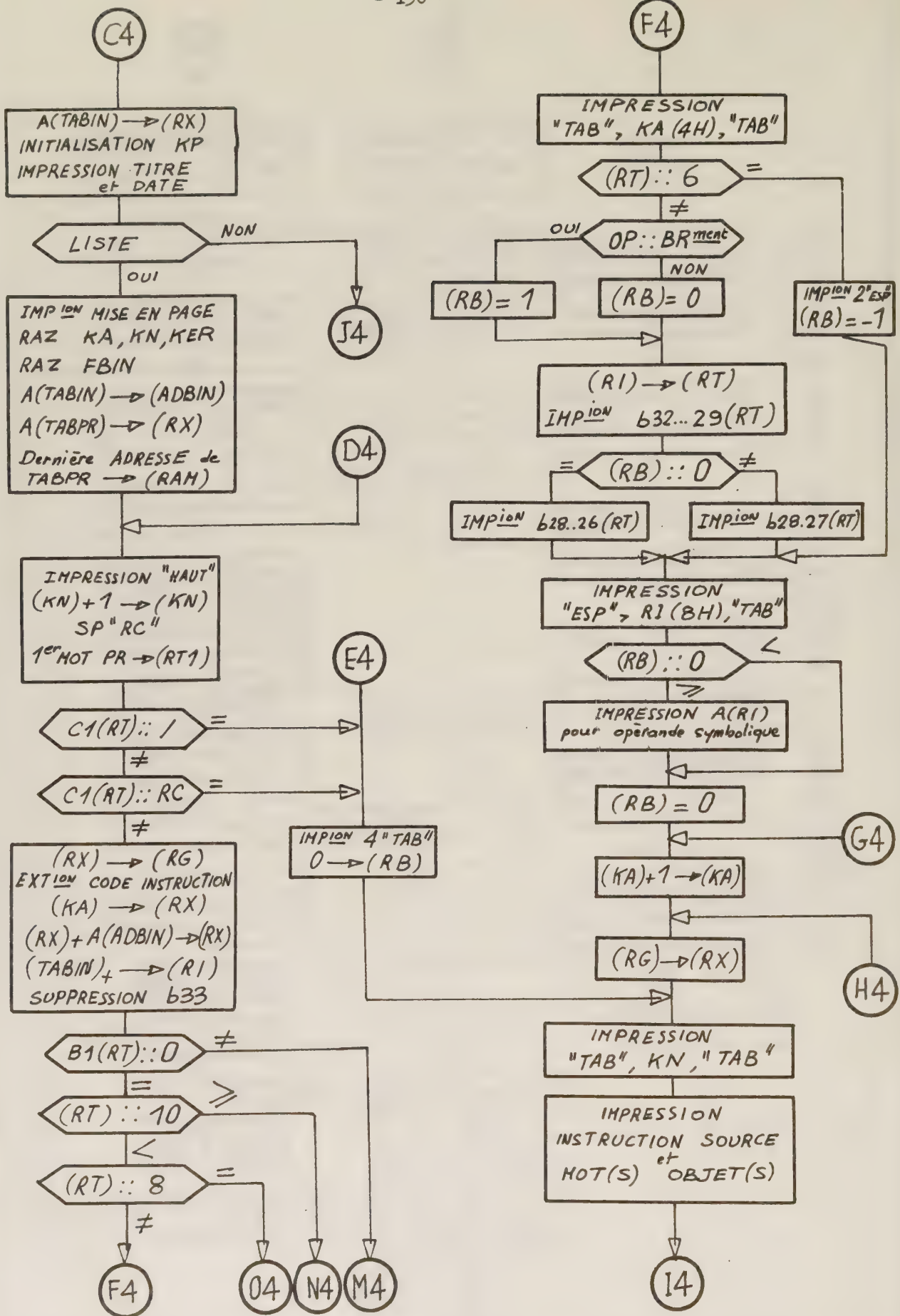


ECLATEMENT des TABLES de SYMBOLES.

TABLE des SYMBOLES
TABLE des DEFINITIONS EXTERNES
TABLE des REFERENCES EXTERNES

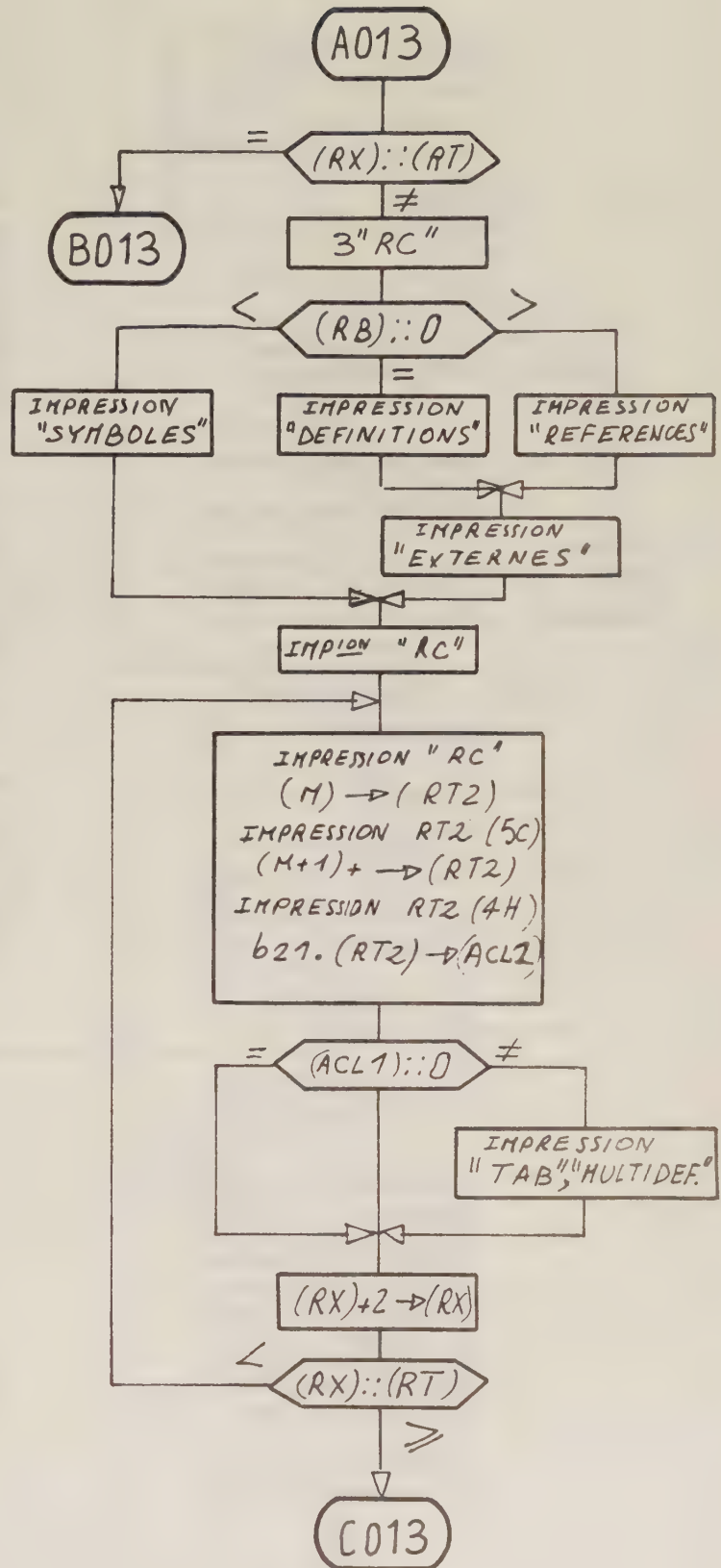
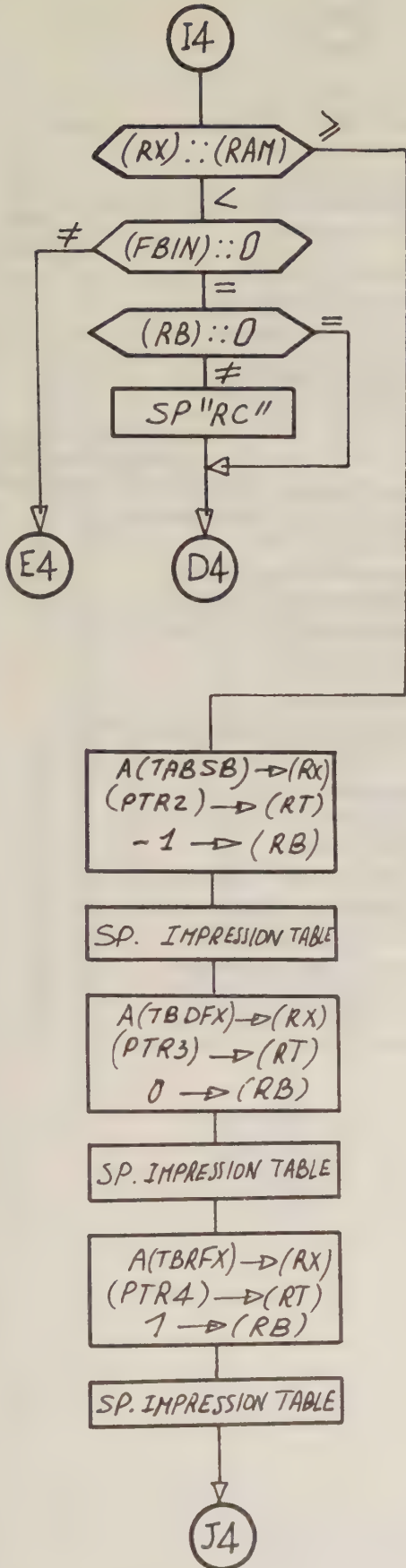


CLASSEMENT des TABLES de SYMBOLES SP. CLASSEMENT ALPHABETIQUE (SP12)



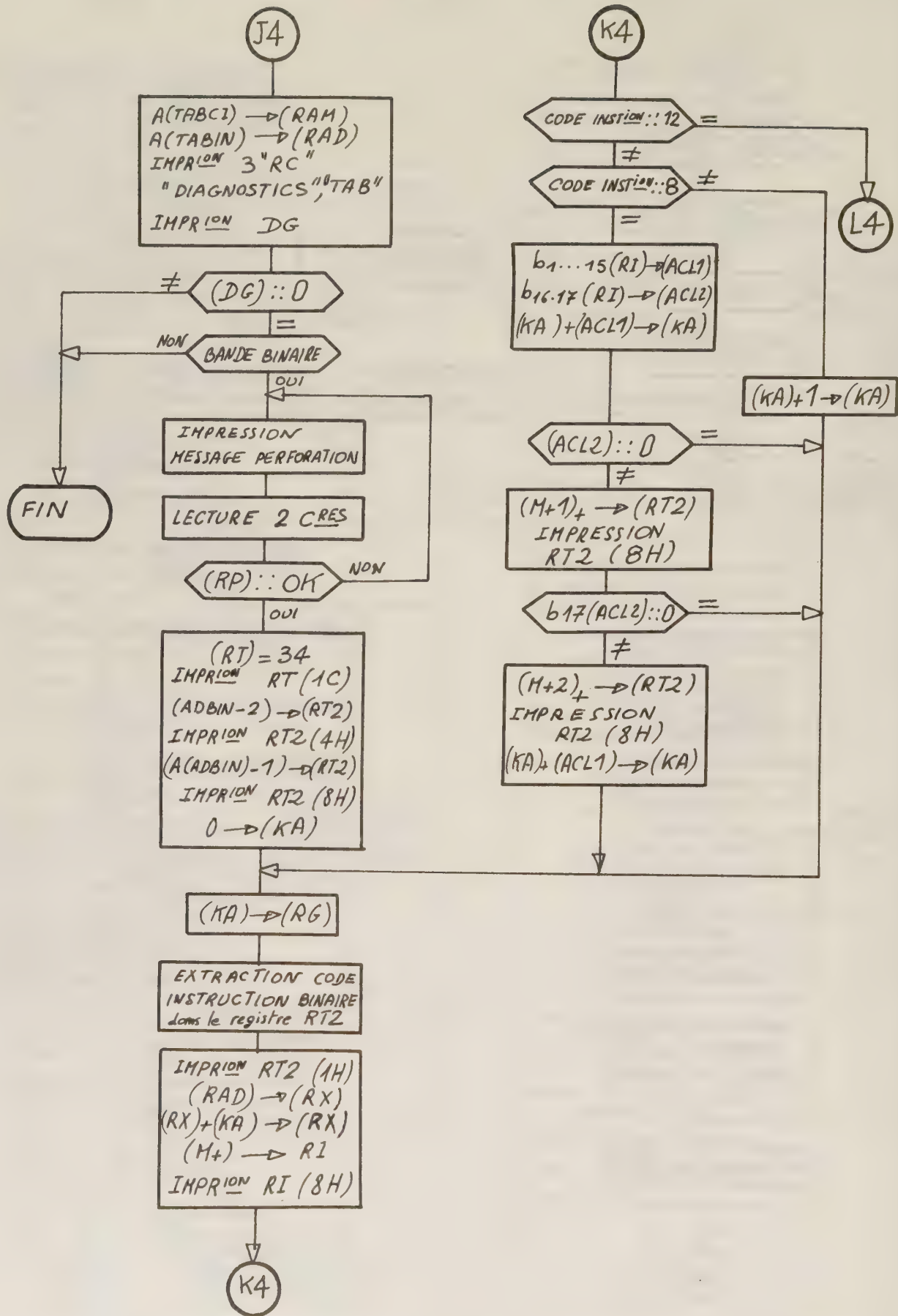
IMPRESSION de la LISTE d'ASSEMBLAGE.

19-ASSEMBLEUR P4 - ORG. Machine (3)



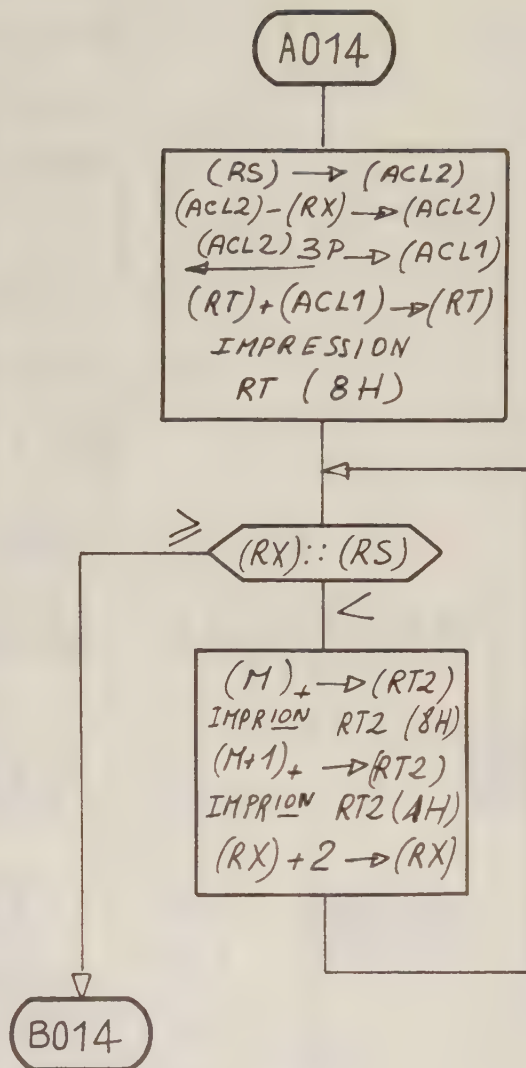
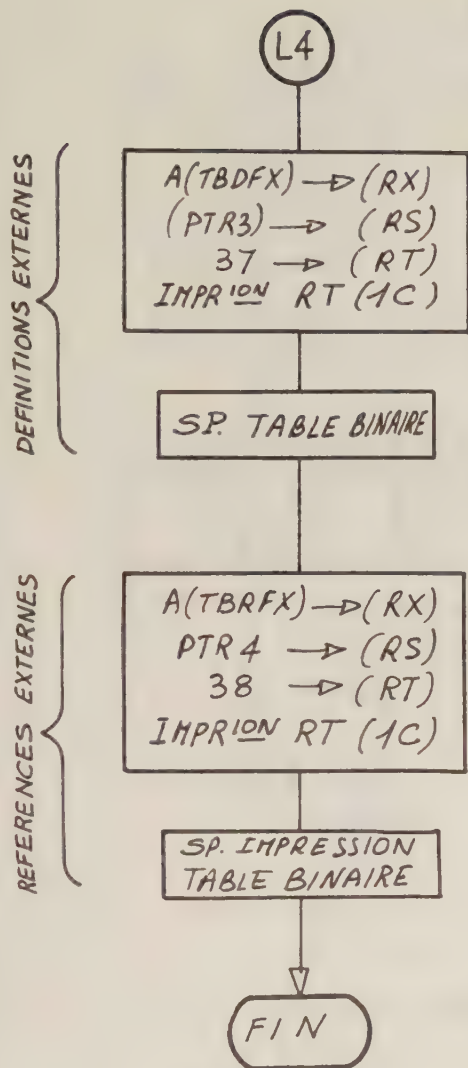
IMPRESSION des TABLES de SYMBOLES

SP. IMPRESSION de TABLE (SP13)



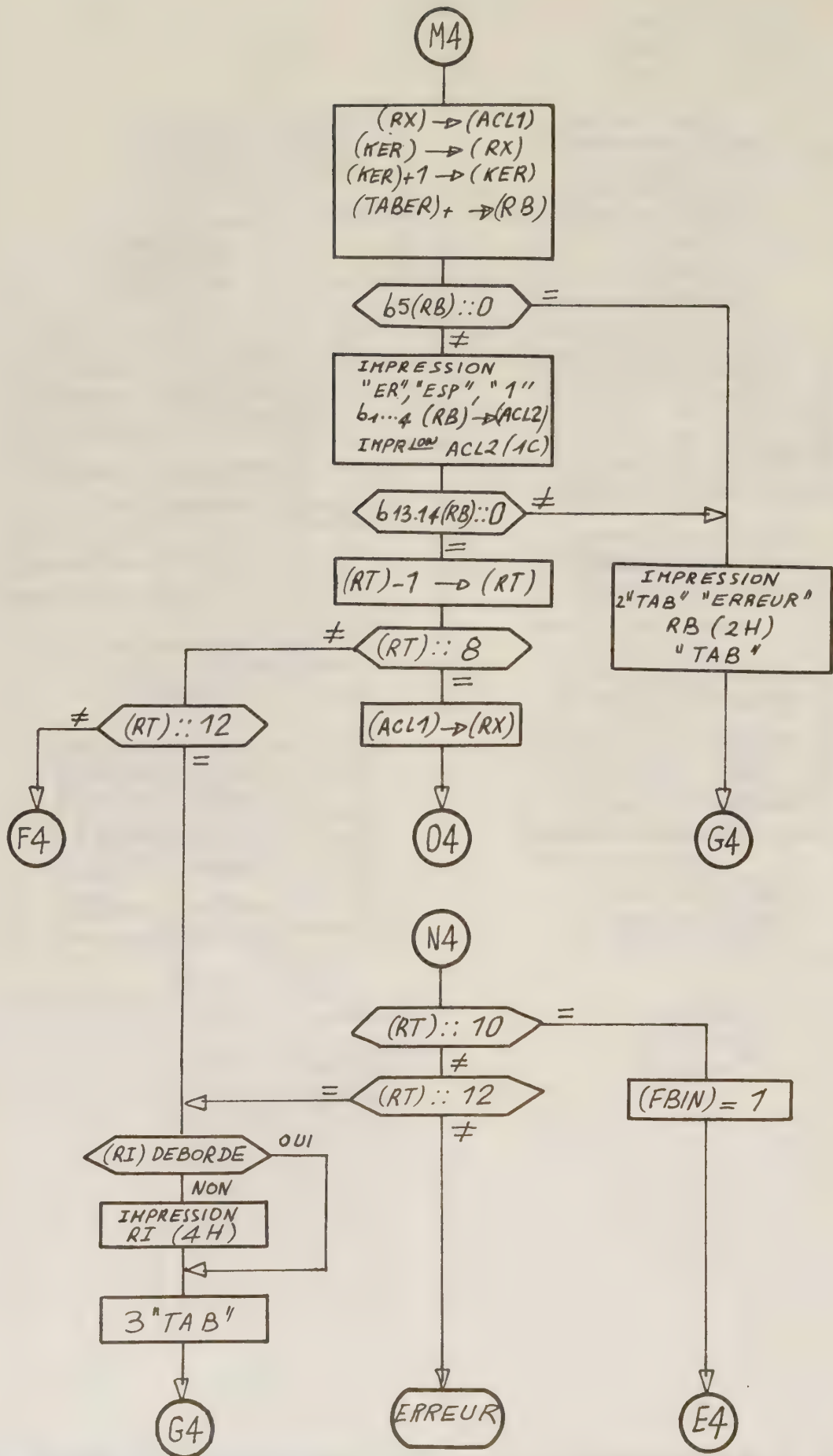
GENERATION BANDE BINAIRE.

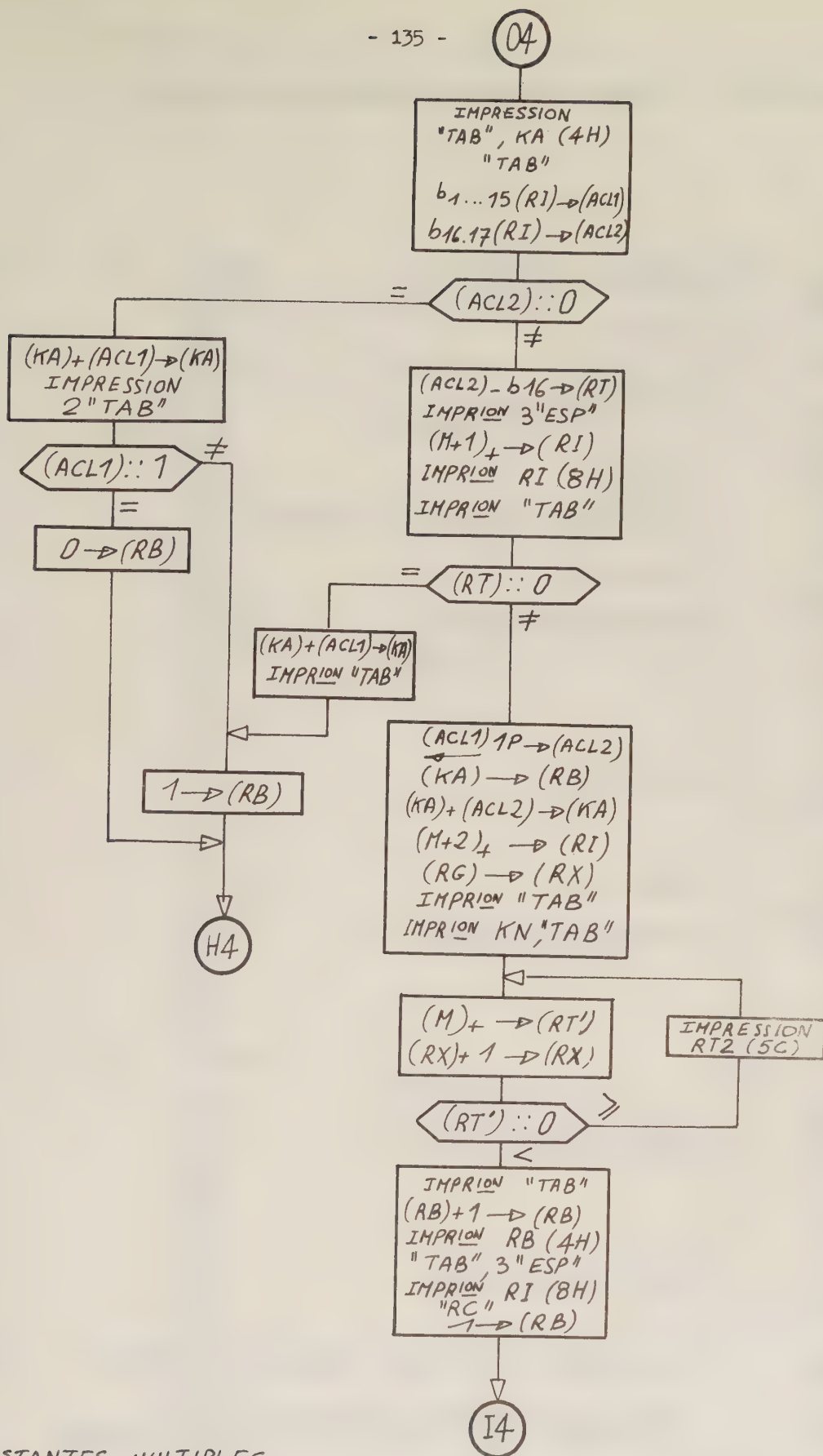
21-ASSEMBLEUR P4 - ORG. Machine (5)



TRAITEMENT des TABLES BINAIRES.

SP. IMPRESSION TABLE BINAIRE (SP14)





CONSTANTES MULTIPLES

24-ASSEMBLEUR P4-ORG. Machine (8). LISTE d'ASSEMBLAGE

ASSEMBLEUR P4 : LISTING ET GENERATION RUBAN OBJET BINAIRE

0	A2EM0Δ	A (TABSB)	(31Δ P1)
	A3EM2ΔS		
2	X	A (TBDFX)	
	A4EM4ΔS		
4	X	A (TBRFX)	
	V11QR2	TEST PTR2::PTRS	
	A5EV1Z14		
	RV8(ΔZ14	A (LISTE)	
	A5EVO	RAZ[RCT]	
	T2R15	ECLATEMENT TABLE DES SYMBOLES	
	A0EM1+		
	I0TW2	TEST B19	
	RV19ΔZ13		
	A1EM0+	DEFINITION EXTERNE	
	T3R15		
	T1M0+		
	T0M1+		
	A3V2		
	RV30Δ		
19	I0TW1	TEST B18	
	RV23ΔZ13		
	A0W3		
	RV25Δ		
23	I0TW4	TEST B20	
	RV32ΔZ13		
25	A1EM0+	REFERENCE EXTERNE	
	T4R15		
	T1M0+		
	T0M1+		
	A4V2		
30	A5V2		
	RV37Δ		
32	RV37ΔZ5		
	A1EM0+	SYMBOLE NORMAL	
	S15R5		
	T1M0+		
	T0M1+		
37	A2V2		
	V2QR11	TEST FIN DE TABLE	
	RV9ΔY14		
	S2R5		
	A11EM0Δ	CLASSEMENT TABLE DES SYMBOLES - A (TABSB)	(31Δ P1)
	T2R14		
	RCV50Δ	APPEL SP CLASSEMENT ALPHABETIQUE	
	A11V256	CLASSEMENT TABLE DES DEFINITIONS EXTERNES	
	T3R14		
	RCV50Δ		
	A11V256	CLASSEMENT TABLE DES REFERENCES EXTERNES	
	T4R14		
	A12EV80Δ		

50	S13ER11 A13R14 T13R5 S13V3 RR12N13 C5TV1	SP CLASSEMENT ALPHABETIQUE CALCUL TAILLE SORTIE 1
56	S14V2 A6EV0 T11R15 A0EM0+	COMPARAISON
60	A1EM2+ T1R13 S13R0 T1R0P13 RV72AP13 T0M2+ T1M0+ A13EM1+ A1EM3+ T1M1+ T13M3+ A6V1	PERMUTATION
72	A15V2 V15TR14 RV6AY13 RR12Z6 S5V1 V5TV1 RV56AY13 RR12	SORTIE 2 (RCT=0)
80	A15EM8LAS	SORTIE 3
81	X RV441A A10EM1+ EV24 EV25 EV25 EV25 EV25 EV48 EV50 EV47 EV39 EV50 EV33 EV45 EV45 EV37 EV15 EV11	A (TABIN) VERS INITIALISATIONS IMPRESSION TITRE ET DATE

ASSEMBLEUR P4

	RCV727Δ	SP ECRITURE 5 CARACTERES
	EV11	
	EV15	
	EV33	
	EV51	
	EV51	
	EV37	
	EV45	
	EV34	
	EV44	
	EV37	
	EV15	
	EV48	
	EV33	
	EV50	
	EV15	
	EV35	
	EV33	
	EV34	
	EV15	
	EV5	
	EVO	
	EV2	
	EV34	
	EV15	
	EV47	
	EV35	
	EV52	
	EV15	
	EV6	
	EV3	
	EV15	
	EV3	
	T5M712Δ	
	A(EV5	
135	EV30	
	SOV1	
	RV135ΔP()	
	T15R()	
	A15EM14()ΔS	
140	X	A (DATE)
	IEOM0+	
	EV11	
	IEOM1+	
	EV11	
	IEOM2+	
	TOR15	
	EV30	
	EV1	
	EV25	

EV25
EV25
S5M0A
RV476AN5
EV30
EV37
EV45
EV48
EV44
EV23
EV10
EV24
EV30
EV35
EV47
EV36
EV37
EV15
EV47
EV34
EV42
EV37
EV52
EV30
EV33
EV36
EV50
EV1
EV30
EV46
EV47
EV23
EV10
EV24
EV48
EV51
EV30
EV30
EV48
EV50
EV47
EV39
EV50
EV33
EV45
EV45
EV37
EV15
EV51
EV47

TEST MEMOIRE SYSTEME
VERS LISTE BINAIRE
IMPRESSION MISE EN PAGE LISTE D'ASSEMBLAGE

CHARGEUR (55Δ P1)

ASSEMBLEUR P4

	EV53		
	EV50		
	EV35		
	EV37		
	EV25		
	EV25		
	A0EVO	RAZ, KA, KN, KER ET MISE EN	
	T2EM464Δ	GARDE PTR2, PTR3	
	T3EM469Δ		
	A3EV8		
	A5EVO	RAZ FBIN	
	T5M341Δ		
	A15V2		
	T15M248Δ		
	A15EM0Δ	A (TABPR)	(3Δ P1)
	A7EM0Δ	A (DERNIERE CELLULE TABPR)	(537Δ P2)
216	EV24	D4	
	A5V1		
	RCV690Δ	SP "RC"	
	A9EM0+	APPEL MOT PROGRAMME	
	I9QW8064	TESTS / ET RC	
	V14TW2048		
	V14TW3200Y13		
	RV230ΔY13		
224	EV30	E4	
	EV30		
	EV30		
	EV30		
	A8EVO		
	RV305Δ		
230	T15R1	EXTRACTION CODE INSTRUCTION	
	T2R11		
	I11TV7		
	C11QV3		
	T13R15		
	A9EV15		
	K9QM346Δ+	A (TABDC)	
	T14R9		
	T11R15		
	A15M240ΔS		
240	X	A (TABCI)	
	I9QM0+		
	A15EV8		
	S15R11		
	C14TM346Δ+	A (TABDC)	
	T14R9		
	T2R15	APPEL INSTRUCTION BINAIRE	
	A15M248ΔS		
248	X	BASE BINAIRE	
	A6EM0+		

	S6W32768N6	SUPPRESSION B33
	I9TV1	TEST B1
	RV357ΔY13	A (M4)
	S14V10	TEST (RT)::10
	RV445ΔP14	A (N4)
	V9TV8	TEST (RT)::8
	RV385ΔZ13	A (O4)
257	EV30	F4
	T2R10	
	RCV746Δ	SP "4H"
	EV30	
	V9TV6	TEST (RT)::6
	EV15Z13	
	EV15Z13	
	S8EV1Z13	
	RV283ΔZ13	
	I6QW15872	TEST CODE OPERATION
	V14TW9216	
	A8EVOY13	
	A8EV1Z13	
	T6R9	IMPRESSION CODE OPERATION
	C9QV28	
	S9V10	
	A9V23P9	
	A9V10	
	ER9	
	C14TV27	
	I14TV14Z8	
	I14TV12Y8	
	S13V10	
	A13V23P13	
	A13V10	
	ER13	
283	EV15	IMPRESSION CODE OBJET
	T6R10	
	RCV748Δ	SP "8H"
	EV30	
	RV302ΔN8	
	V10QV33	IMPRESSION ADRESSE MEMOIRE
	RV297ΔZ14	
	I6QV12	
	V14TV4	
	RV297ΔZ13	
	I6QW32512	
	V14TW13056	
	V14TW21248Y13	
	RV302ΔY13	
297	C6TV4	
	C6TV7Y8	
	I6TV32767	

	T13R10	
	RCV746Δ	SP "4H"
302	A8EV0	
303	A2V1	G4
304	T1R15	H4
	EV30	
	IE5R5	IMPRESSION (KN)
	EV30	
308	A10EM0+	IMPRESSION PROGRAMME SOURCE
	A15V1	
	RV313ΔN10	
	A12EV308Δ	
	RV727Δ	SP "5C"
313	I10TV32767	IMPRESSION INSTRUCTIONS BINAIRES
	RV337ΔZ13	
	T15R1	
316	T2R15	
	A15M248Δ	
	A6EM0+	
	S6W32768N6	
	EV30	
	T2R10	
	RCV746Δ	SP "4H"
	T6R10	
	EV30	
	EV15	
	EV15	
	EV15	
	RCV748Δ	SP "8H"
	T13R15	
	RCV690Δ	SP "RC"
	A2V1	
	S15V1	
	T15R13	
	S15V2	
	RV316ΔP14	
	T1R15	
337	T15R14	I4 - TEST FIN DE PROGRAMME
	S14R7	
	RV462ΔP14	A (TRAITEMENT TABLES DES SYMBOLES)
	A13EM341ΔS	
341	X	FBIN
	RV224ΔY13	A (E4)
	A12EV216Δ	A (D4)
	RV690ΔY8	SP "RC"
	RR12	
346	1	TABDC - B1
	16	B5
	256	B9
	4096	B13

	65536	B17	
	1048576	B21	
	16777216	B25	
	268435456	B29	
	X	B33	
	F	RESERVE	
	F		
357	T15R13	M4, TRAITEMENT ERREURS	
	T0R15		
	A(V1		
	A8EMXΔ+	A (TABER)	(135 Δ P3)
	I8QV16	TEST B5	
	RV376ΔZ14	VERS ERREUR 2	
	EV37	ERREUR 1	
	EV50		
	EV15		
	I8QV15		
	EV1		
	ER14		
	I8QV12288	TEST B13-B14	
	RV376ΔY14	A (ERREUR 2)	
	S9V1		
	V9QV8		
	RV459ΔY14		
	T13R15		
	RV385Δ		
376	EV30	ERREUR 2	
	EV30		
	RCV(XΔ		
	C8TV8		
	T8R10		
	RCV744Δ	SP "2H"	
	EV30		
	X	RESERVE	
	RV303Δ	A (G4)	
385	EV30	O4.TRAITEMENT CONSTANTES	
	T2R10		
	RCV746Δ		
	EV30		
	I6TV32767	B1,...15 (RI) → ACL1	
	I6QV98304	B16,B17 (RI) → ACL2	
	RV399ΔY14		
	A2R13	TRAITEMENT "RESERVATION"	
	EV30		
	EV30		
	V13QV1		
	A8EV0Z14		
	A8EV1Y14		
	RV304Δ	A (H4)	
399	T14R9	IMPRESSION CONSTANTE	

	EV15	
	EV15	
	EV15	
	A6EM1+	
	S6W32768N6	
	T6R10	
	RCV748Δ	SP "8H"
	S9W32768	TEST RT
	EV30	
	RV413ΔY9	
	A2R13	
	A8EV1	
	RV304Δ	A (H4)
413	K13QV1	IMPRESSION PROGRAMME SOURCE
	T2R8	
	A2R14	
	A6EM2+	
	S6W32768N6	
	T1R15	
	EV30	
	IE5R5	
	EV30	
422	A10EM0+	
	A15V1	
	RV427ΔN10	
	A12EV422Δ	
	RV727Δ	SP "5C"
427	EV30	IMPRESSION 2ème PARTIE CONSTANTE
	A8V1	
	T8R10	
	RCV746Δ	SP "4H"
	EV30	
	EV15	
	EV15	
	EV15	
	T6R10	
	RCV748Δ	SP "8H"
	RCV690Δ	SP "RC"
	A8EV1	
	RV337Δ	A (I4)
	F	RESERVE
	A5EV1	
	A1EV0	
	RV83Δ	
	F	RESERVE
445	RV449ΔY14	N4 - TRAITEMENT CODES SPECIAUX
	A14EV1	
	T14M341Δ	FBIN = 1
	RV224Δ	A (E4)
	V9QV12	TEST (RT)::12

	FY14	ERREUR	
451	EV30		
	RV455ΔD6	TEST DEBORDEMENT	
	T6R10		
	RCV746Δ	SP "4H"	
455	EV30		
	EV30		
	EV30		
	RV303Δ	A (G4)	
459	V9QV12	TEST (RT)::12	
	RV451ΔZ14		
	RV257Δ	A (F4)	
462	A15EM0Δ	A (TABS B) - TRAITEMENT DES TABLES DES	(31Δ P1)
	A5EM464ΔS	SYMBLES	
464	X	PTR2	
	S8EV1		
	RCV615Δ	SP13 - IMPRESSION TABLE	
	A15EM2Δ	A (TBDFX)	
	A5EM469ΔS		
469	X	PTR3	
	A8EV0		
	RCV615Δ	SP13	
	A15EM4Δ	A (TBRFX)	
	T4R5		
	A8EV1		
	RCV615Δ	SP13	
	A7EM240Δ	A (TABCI) - IMPRESSION LISTE BINAIRE	
	A8EM248Δ	A (BASE BINAIRE)	
	EV25	MESSAGE : "DIAGNOSTIC"	
	EV25		
	EV25		
	EV24		
	EV36		
	EV41		
	EV33		
	EV39		
	EV46		
	EV47		
	EV51		
	EV52		
	EV41		
	EV35		
	EV30		
	A1EM0Δ	IMPRESSION (DG)	(62 Δ P3)
	IEOR1		
	EV25		
	FY1	FIN SI (DG) ≠ 0	
	A10EM0Δ	A (MEMOIRE SYSTEME)	CHARGEUR (55 Δ P1)
	FZ10		
499	EV47		

	EV50	MESSAGE : ORGANE DE PERFORATION ?
	EV39	
	EV33	
	EV46	
	EV37	
	EV15	
	EV36	
	EV37	
	EV15	
	EV48	
	EV37	
	EV50	
	EV38	
	EV47	
	EV50	
	EV33	
	EV52	
	EV41	
	EV47	
	EV46	
	EV23	
	EV39	
	EV24	
	EV25	
	A13EL0S	LECTURE 2 CARACTERES
	X	
	K13QV6	
	A14L0S	
	X	
	V14TV3051	TEST "OK"
	RV499ΔY13	
	A9EV1(X)	PERFORATION "AVANCE-BANDE"
532	EM533ΔS	
533	32	
	S9V1	
	RV532ΔP9	
	EV34	PERFORATION CODE B (34)
	T8R15	
	S15V2	
	A10EM0+	PERFORATION NOMBRE D'INSTRUCTIONS
	RCV746Δ	SP "4H"
	A10EM1+	PERFORATION NOM PROGRAMME
	RCV748Δ	SP "8H"
	A2EV0	RAZ (KA)
544	T2R1	EXTRACTION CODE INSTRUCTION
	I1QV7	
	C1TV3	
	T14R15	
	A9EV15	
	K9TM346Δ+	A (TABDC)

	T13R9	
	T1R15	
	A15R7	
	I9TM0+	
	A15EV8	
	S15R14	
	C13QM346Δ+	A (TABDC)
	T13R10	PERFORATION CODE INSTRUCTION (HEXADECIMAL)
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T8R15	PERFORATION INSTRUCTION
	A15R2	
	A6EM0+	
	S6W 32768N6	
	T6R10	
	RCV748Δ	SP "8H"
	V13QV12	TEST CODE::12 (K4)
	RV587ΔZ14	A (L4)
	V13QV8	TEST CODE::8
	A2V1Y14	
	RV544ΔY14	
	I6TV 32767	EXTRACTION B1...15 (RI)
	A2R13	
	I6QV98304	EXTRACTION B16,17 (RI)
	RV544ΔZ14	
	A10EM1+	PERFORATION CONSTANTE
	S10W 32768N10	
	RCV748Δ	SP "8H"
	I6QV65536	
	RV544ΔZ14	
	A10EM2+	PERFORATION 2ème PARTIE CONSTANTE D.1
	S10W 32768N10	
	RCV748Δ	SP "8H"
	A2R13	
	RV544Δ	
587	A15EM2Δ	(L4) - TABLE DEFINITIONS EXTERNES
	EV37	
	A5EM469Δ	A (PTR3)
	RCV601Δ	SP "TABLE BINAIRE"
	A15EM4Δ	TABLE REFERENCES EXTERNES
	EV38	
	T4R5	PTR4
	RCV601Δ	SP "TABLE BINAIRE"
	A1EV1(X)	
596	EM597ΔS	PERFORATION "AVANCE-BANDE"
597	32	
	S1V1	
	RV596ΔP1	

ASSEMBLEUR P4

601	F T5R14 S14R15 T14R10 T12R9 RCV744Δ	FIN DU TRAITEMENT SP TABLE BINAIRE
606	V15TR5 RR9Z13 A10EM0+ RCV748Δ A10EM1+ RCV746Δ A15V2 RV606Δ	PERFORATION TAILLE DE LA TABLE SP "2H" TEST FIN DE TABLE SORTIE PERFORATION SYMBOLE SP "8H" PERFORATION ADRESSE SP "4H"
615	F V15QR5 RR12Z14 T12R7 RCV690Δ RCV690Δ RCV690Δ EV24 RV632ΔP8 EV51 EV57 EV45 EV34 EV47 EV44 EV37 EV51 RV664Δ	RESERVE SP IMPRESSION TABLE (A013) SORTIE 1 GARAGE ADRESSE RETOUR SP "RC" SP "RC" SP "RC" HAUT
632	RV644ΔZ8 EV50 EV37 EV38 EV37 EV50 EV37 EV46 EV35 EV37 EV51 RV655Δ	IMPRESSION : "SYMBOLES"
644	EV36 EV37 EV38 EV41 EV46 EV41	IMPRESSION : "REFERENCES"
		IMPRESSION : "DEFINITIONS"

	EV52	
	EV41	
	EV47	
	EV46	
	EV51	
655	EV15	IMPRESSION : "EXTERNES"
	EV37	
	EV56	
	EV52	
	EV37	
	EV50	
	EV46	
	EV37	
	EV51	
664	RCV69(X)	SP "RC"
665	RCV69(X)	SP "RC"
	A1(EM)+	IMPRESSION SYMBOLE
	RCV727Δ	SP "5C"
	EV30	IMPRESSION ADRESSE
	A1(EM)+	
	I10TW8	TEST B21
	RCV746Δ	SP "4H"
	RV685ΔZ13	
	EV30	MESSAGE : "MULTIDEF."
	EV45	
	EV53	
	EV44	
	EV52	
	EV41	
	EV36	
	EV37	
	EV38	
	EV23	
	EV10	
	EV24	
	A15V2	
	V15TR5	TEST FIN DE TABLE
	RV665ΔY13	
	RCV69(X)	SP "RC"
	RR7	SORTIE
690	EV25	SOUS-PROGRAMME "RC" ET SAUT
	A3V1	DE PAGE
	T12R6	
	S12V614Δ	
	RV699ΔN12	
	S3V43	
	RV703ΔP3	
	A3V43	
	RR6	
	S3V46	

	RV703AP3	
	A3V46	
	RR6	
703	A3V3	
704	EV25	SAUT DE PAGE
	S3V1	
	RV704AP3	
	A3EV13	
708	EV30	
	S3V1	
	RV708AP3	
	A3EM712AS	
712	X	KP (COMPTEUR DE PAGES)
	A3V1	
	IEOR3	IMPRESSION N° DE PAGE
	T3M712A	
	A3EV1	
	EV25	
	EV25	
	EV25	
	RR6	
721	K10QV19	SP ECRITURE CARACTERES ALPHANUMERIQUES
	RV737A	
	K10QV13	(3C)
	RV734A	
	K10QV7	(4C)
	RV731A	
727	K10QV1	(5C)
	T14R10	
	C10QV25	
	ER10	
731	T14R10	
	C10QV25	
	ER10	
734	T14R10	
	C10QV25	
	ER10	
737	T14R10	
	C10QV25	
	ER10	
	T14R10	
	C10QV25	
	ER10	
	RR12	SORTIE
744	K10QV23	SP ECRITURE CARACTERES HEXADECIMAUX
	RV783A	
746	K10QV15	(4H)
	RV771A	
748	C10QV28	(8H)
	S10V10	

	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
771	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
783	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	RR12	
	F	SORTIE
	F	PLACE RESERVEE
	F	
	F	
	FE	FIN DE BANDE

IX - DIRECTIVES

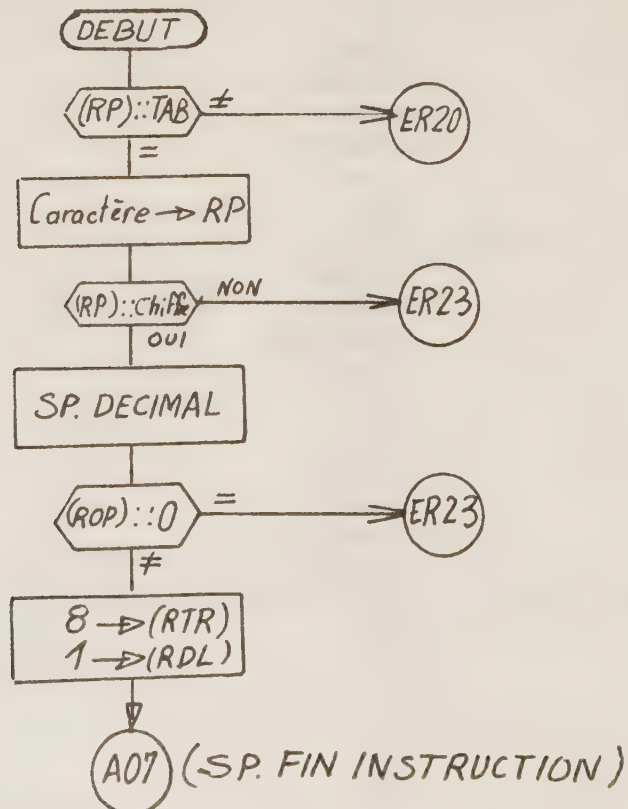
Nous avons vu le rôle des directives et leurs syntaxes au chapitre précédent. Voyons maintenant la façon dont elles sont réalisées.

1 - Réservation (RES) (organigramme 22)

L'opérande de la pseudo-instruction est un nombre décimal qui indique la place à réserver en mémoire. Après test du caractère de séparation, le caractère suivant est analysé ; celui-ci doit être un chiffre et le sous-programme de lecture décimale assure le chargement de l'opérande en registre ROP.

Après vérification de la validité du nombre, les registres RTR et RDL sont respectivement positionnés aux valeurs 8 et 1 (code instruction et longueur du mot binaire généré).

C'est dans le sous-programme "fin d'instruction", que le code et l'instruction binaire sont chargés en mémoire et que le compteur d'adresse évolue de la quantité définie par la directive.



2 - Fin de programme (FIN)

La pseudo-instruction "FIN" indique à l'assembleur que le programme est terminé (éventuellement il pourrait s'agir de la fin d'une macro-instruction).

L'absence d'étiquette est obligatoire ; après ce contrôle, le premier élément de la table des symboles est placé dans un registre (ROP1) si celui-ci définit le nom du programme (valeur correspondante du compteur d'adresse nulle). Le programme teste ensuite la présence d'un opérande qui définit l'adresse d'exécution du programme.

L'absence d'opérande entraîne la mise en place des digits B32 et B33 dans le registre d'instruction (repère conventionnel indiquant l'absence d'adresse d'exécution).

Dans le cas contraire, le symbole ou la constante sont analysés et l'adresse est placée dans le registre RI (la présence du digit B18 rappelle que l'adresse n'est pas translatable : cas d'une constante).

Si le programme est effectivement terminé le nombre d'instructions et le nom symbolique du programme (le mot est rempli par des espaces s'il n'existe pas) sont respectivement chargés au 1er et au 2ème emplacement de la table des instructions binaires ; la commande est ensuite passée au SP "fin d'instruction" (B07).

3 - Génération de chaîne de caractères (TEXTE)

La directive "TEXTE" génère une chaîne de caractères qui sont placés dans les mots-machine successifs.

Après contrôle du caractère de séparation et initialisation des principaux registres indicateurs : RSL (indicateur simple-longueur), KM (compteur de mots générés), RTR, RI et RC (compteur de caractères par mot)

les opérations débutent par la lecture du caractère et par le test "retour de chariot" qui permet de détecter la fin de la chaîne alphanumérique.

Si le caractère n'est pas un retour de chariot, il est placé dans le mot en cours de formation ; quand celui-ci est complet (RC = 5), le programme se débranche vers la "fin d'instruction" qui place le mot et le code instruction (6 pour constante) dans leurs tables respectives, sinon on passe à l'analyse du caractère suivant. Quand la fin de l'expression est reconnue, le dernier mot est éventuellement complété à cinq caractères par des espaces (code interne "15") et s'il y a plus d'un mot généré le nombre correspondant (KM) est placé dans le mot de séparation du programme source en vue du listing (cf. paragraphe VIII 1 b). La séquence s'oriente ensuite vers la fin d'instruction.

4 - Définition de constantes (DFC)

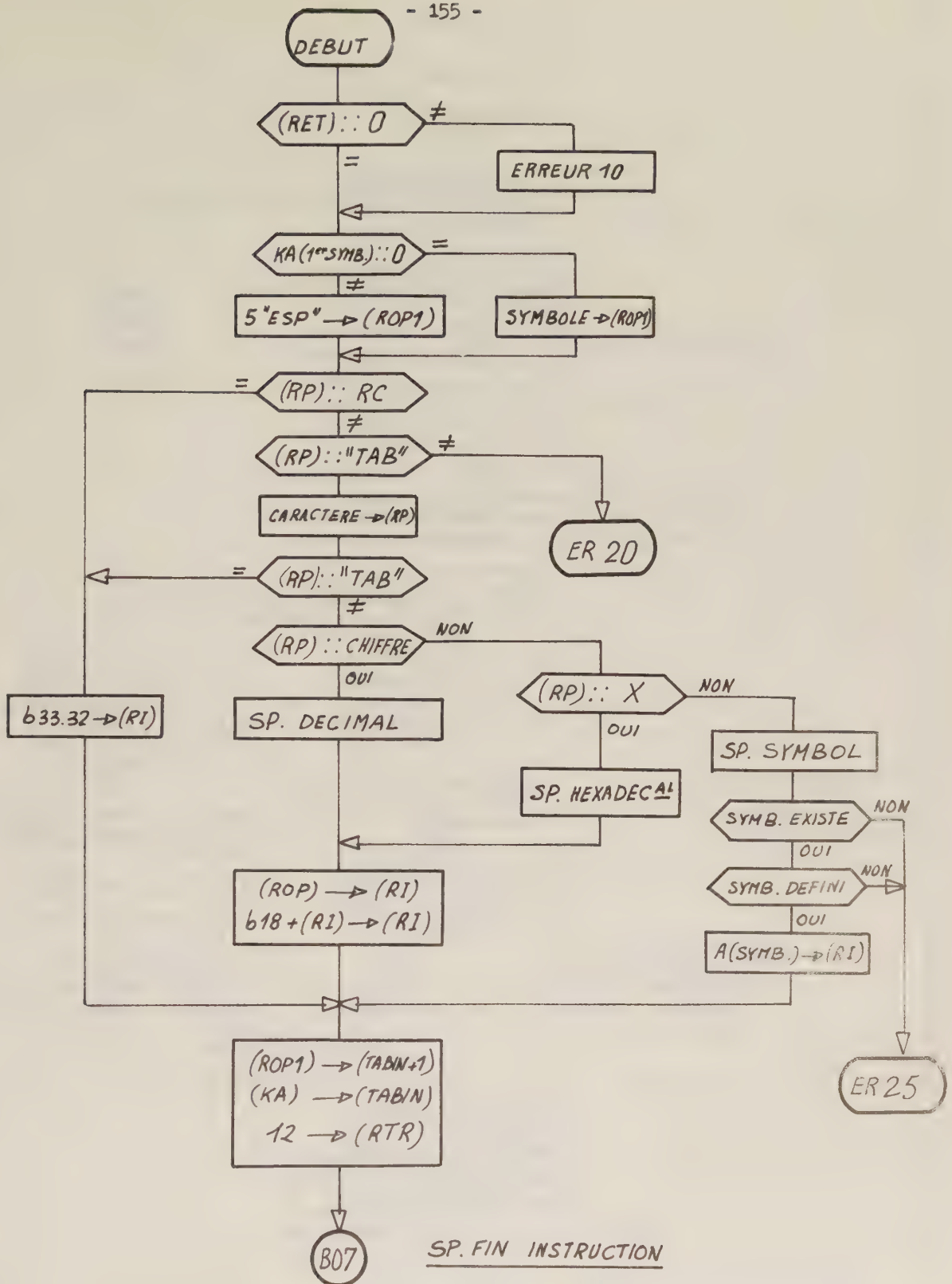
Afin de ne pas alourdir l'exposé, nous ne décrivons pas en détail le traitement effectué par cette directive.

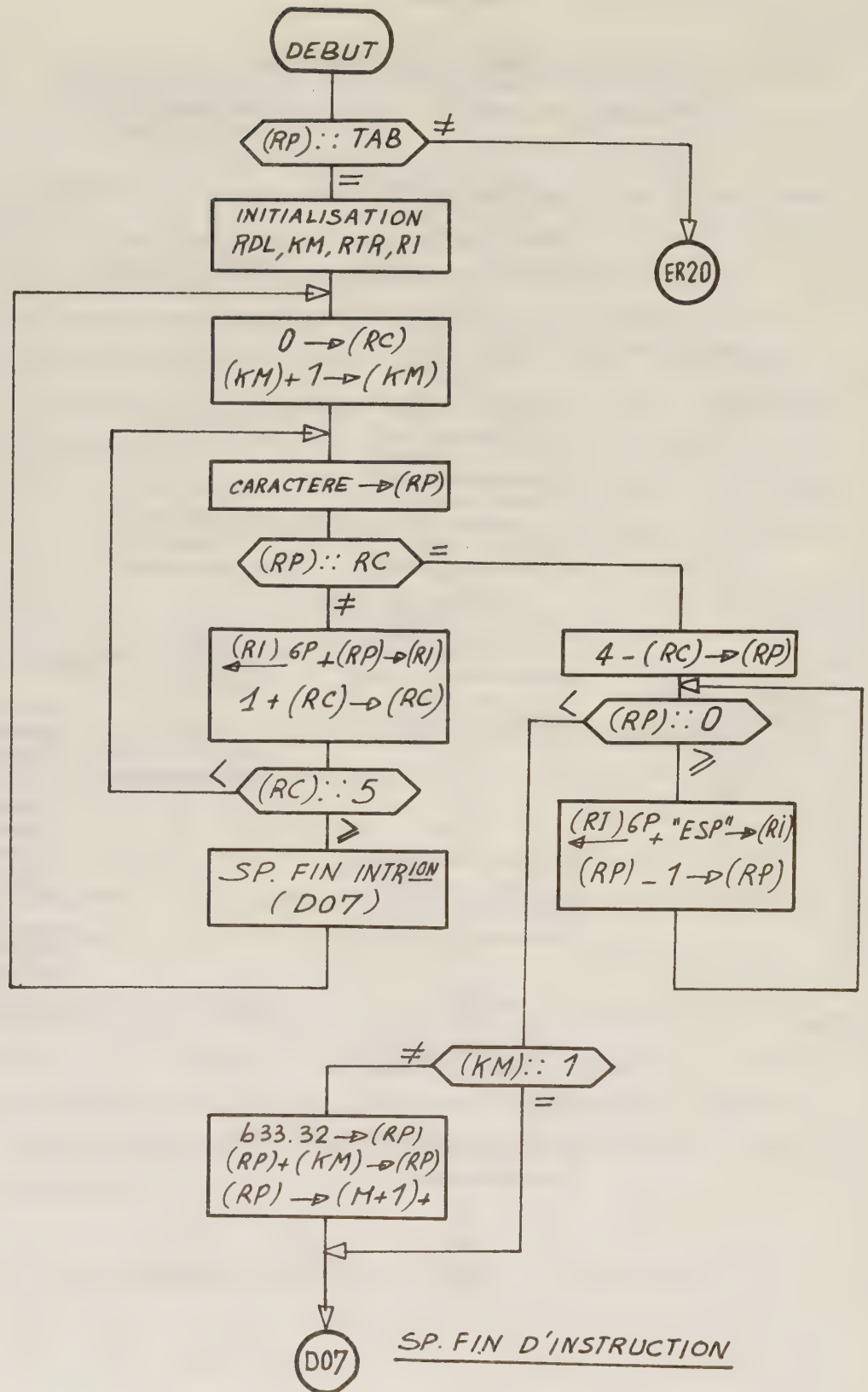
Les problèmes posés sont ceux des conversions entre les nombres décimaux ou hexadécimaux fournis et les diverses représentations binaires (virgule fixe et virgule flottante normalisées).

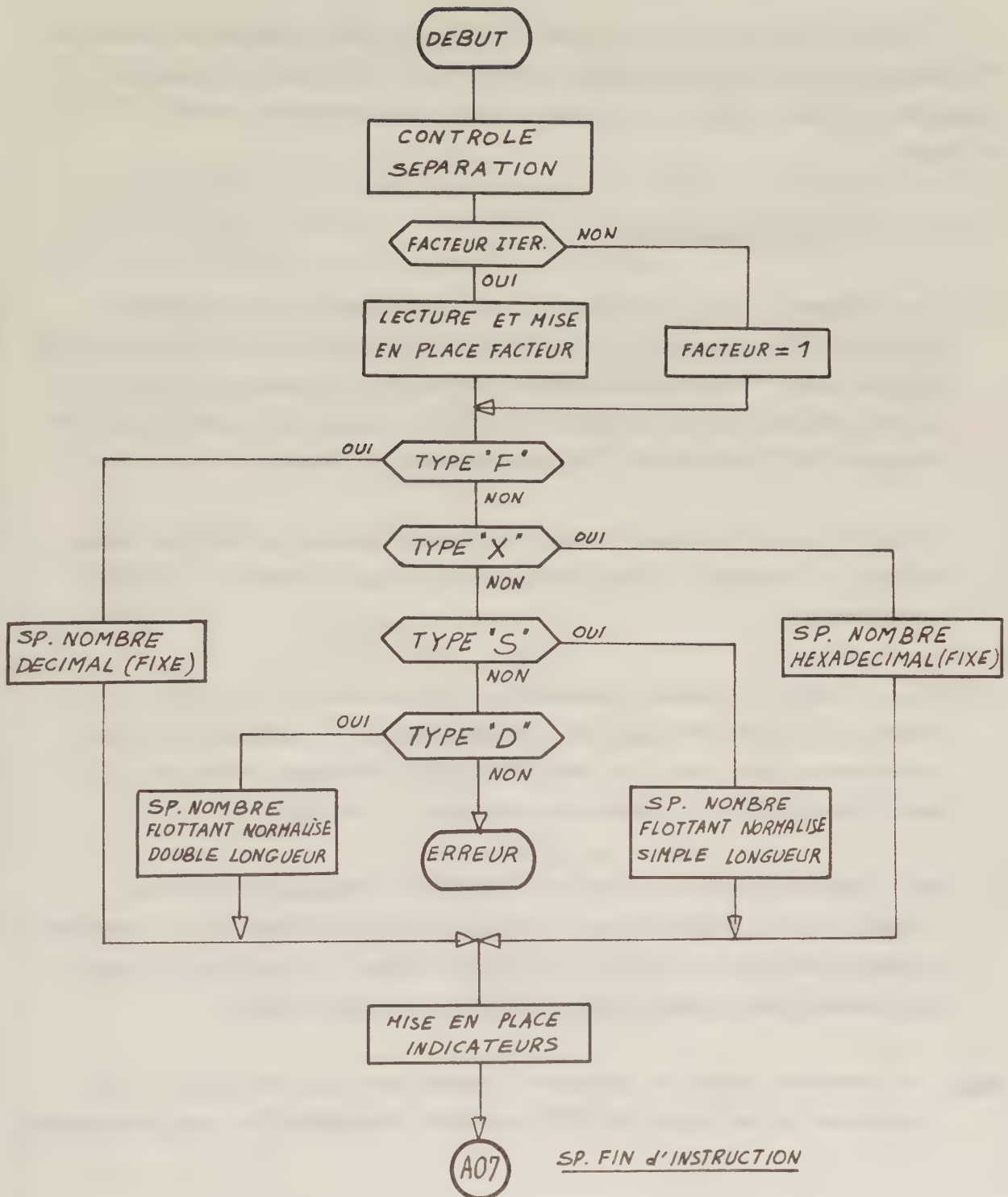
Après contrôle du cliché de l'instruction, la présence d'un facteur itératif est testée. S'il existe, il est traité et mis en garde. La suite du traitement consiste à s'orienter vers le sous-programme qui traduira le nombre dans la convention désirée compte tenu du facteur d'échelle éventuel (binaire ou décimal).

Quand la constante est générée, les indicateurs de type et de longueur sont positionnés et le programme se débranche vers le SP "fin d'instruction" qui, ici encore, assure le chargement des instructions et codes correspondants.

Nota : les programmes concernant ces directives sont donnés en annexe .







X - SOUS-PROGRAMMES (Organigrammes en annexe)

Nous décrivons ici les divers sous-programmes utilisés au cours de l'assemblage. Ils sont généralement placés dans l'une des sections de programme qui les utilise, les organigrammes correspondants sont en annexe.

1 - Liaisons externes

- a) - Les liaisons entre programmes et sous-programmes sont normalement assurées par le registre 12. Ce registre a en effet un rôle particulier puisque dans l'instruction machine CAB500 de branchement et retour, la valeur actuelle du compteur ordinal est rangée en registre 12 (adresse de l'instruction géographiquement suivante).

Il suffit alors de placer en fin du sous-programme un ordre de branchement à l'adresse contenue dans R12 pour que le retour s'effectue normalement.

- b) - Dans le cas où le sous-programme en utilise un autre, ou si le registre 12 doit être employé pour des calculs, l'adresse de retour est mise en garde dans une mémoire du sous-programme principal qui sera l'opérande de l'instruction assurant la sortie.
- c) - Si l'adresse de retour n'est pas l'adresse normalement suivante, l'appel au sous-programme est réalisé par une instruction de branchement normal (sans mise en garde du compteur ordinal) le registre 12 ayant été préalablement chargé par l'adresse de retour voulue.

Nota : on trouvera, dans les sections d'assembleur déjà rencontrées, des exemples de ces types de liaisons entre programmes et sous-programmes.

2 - SP "Lecture Symbole" (SP1)

Il est constitué de la répétition de la séquence de lecture d'un caractère et de son chargement dans le registre opérande (ROP) par décalage et addition.

Les symboles ou codes opération traités par ce sous-programme ont au maximum cinq caractères ; si en fin de séquence, le dernier caractère n'est pas une séparation, le symbole est illégal.

Les caractères alphanumériques utilisables ont un code binaire décimal compris, entre 0 et 9 pour les chiffres et entre 33 et 58 pour les lettres. Afin de réduire le nombre d'instructions du test de sortie au minimum, nous adoptons le procédé représenté par la figure 16.

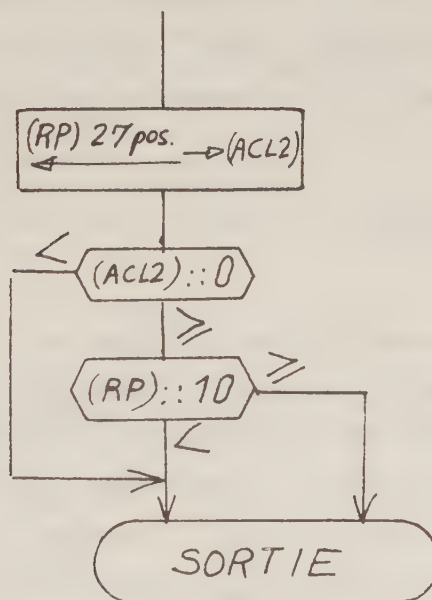


Figure 16 - Test du séparateur

En effet, un décalage de 27 positions à gauche amène le digit B6 en position B33, si le caractère analysé est une lettre (B6 = 1), le contenu du registre devient négatif (B33 = 1). Par ailleurs, si on teste le registre par rapport à la valeur 10, le contenu sera inférieur s'il s'agit d'un chiffre.

Ce test est rédigé en langage autocode CAB500 avec 4 instructions seulement, toutes les autres formes de test plus évidentes a priori en nécessitent davantage.

3 - SP "REGISTRE" (SP2)

Il réalise la lecture d'un nombre décimal de deux chiffres au maximum. Le premier caractère est lu dans le registre (ROP), le second dans (RP) ; si ce dernier n'est pas un chiffre, le nombre cherché est dans (ROP) et l'on sort du sous-programme sinon le contenu de (ROP) est multiplié par 10 et ajouté à (RP) tandis que le caractère suivant est lu pour être analysé ultérieurement.

4 - SP "IMMEDIAT" (SP3)

Le sous-programme assure la mise en place de l'opérande immédiat d'une instruction. Après lecture du premier caractère, le contenu du registre RCI indique la nature de l'instruction et le maximum admissible est positionné à la valeur 33 pour une opération de décalage. S'il s'agit d'une instruction d'un autre type, le premier caractère peut être "X" (facteur d'échelle) et la constante doit alors être cadrée en B17 (valeur immédiate W dans le langage machine).

Si le caractère à tester est un chiffre, la constante est décimale et le sous-programme correspondant intervient, sinon le nombre est hexadécimal et précédé des caractères X =. Après contrôle du caractère de séparation et de la validité de l'opérande lu, la valeur de RCI indique si le traitement est terminé.

Dans le cas des décalages ($RCI \neq 0$), il faut réaliser le transcodage du nombre de positions de décalage (cf. paragraphe VI 2 b) ; nous avons vu, en effet, que dans l'instruction machine c'est un digit particulier qui détermine la grandeur du décalage à effectuer.

Ce digit est obtenu par appel indexé à la table des 2^N représentée par la figure 17.

	32	16	1	
TAB2N			1	1
+1			1	2
+2			1	4
+3			1	8
+4			1	16
+5			1	'
+6			1	'
+7			1	'
+8			1	'
+9			1	'
+10			1	'
+11			1	'
+12			1	'
+13			1	8192
+14			1	16384
+15			1	32768

Figure 17

La valeur de l'index est calculée de la façon suivante :

- pour un décalage à droite $(RX) = 16 - N$
- pour un décalage à gauche $(RX) = N - 16$

- après cette transformation :

si $(RX) > 0$, le code adresse 1 est égal à 6 (W)

si $(RX) \leq 0$, le code d'adresse 1 est 4 (V) et l'index est augmenté du nombre 17.

Ce qui se traduit par :

- décalage à droite :

$$\text{indice V : } (RX) = 16 - N + 17 = 33 - N \quad V = 2^{33-N}$$

$$\text{indice W : } (RX) = 16 - N \quad W = 2^{16-N}$$

- décalage à gauche :

$$\text{indice V : } (RX) = N - 16 + 17 = N + 1 \quad V = 2^{N+1}$$

$$\text{indice W : } (RX) = N - 16 \quad W = 2^{N-16}$$

Ces valeurs du digit de décalage correspondent bien aux conditions définies au paragraphe VI 2 b.

5 - SP "DECIMAL" et "HEXADECIMAL"

Ces deux sous-programmes ont la même structure, ils ont pour but respectif d'assurer la transformation d'un nombre décimal ou hexadécimal en binaire naturel.

Dans les deux cas, si le caractère lu n'est pas un chiffre (décimal ou hexadécimal) on sort du sous-programme.

a) - La conversion de décimal en binaire s'obtient par lecture des caractères successifs et calcul de $\sum 10 C_i + C_{(i+1)}$ C_i étant le caractère courant. En effet un nombre décimal représenté en numération de position par $N = \dots d c b a$ (chiffres lus dans cet ordre sera obtenu en binaire par :

$$\dots 10 (10(10 d + c) + b) + a = \dots 10^3 d + 10^2 c + 10 b + a$$

b) - Pour la conversion d'un nombre hexadécimal la relation est du

$$\text{type } \sum 16 C_i + C_{(i+1)}$$

Remarque :

- Pour un nombre hexadécimal, le facteur de multiplication étant une puissance entière de 2, un décalage de 4 positions suffit.
- Pour un nombre décimal la méthode utilisée est du même genre:

Si la multiplication par le facteur 10 est exécutée directement par l'instruction microprogrammée disponible, le temps moyen est de 60 ms. Cette opération peut être réalisée par une suite de 5 instructions câblées et le temps d'exécution est alors 1,5 ms :

- décalage de 3 positions à gauche : $8 C_i$
- addition au registre contenant le caractère suivant : $8 C_i + C_{(i+1)}$
- décalage de 1 position à gauche : $2 C_i$
- addition au registre précédent : $8 C_i + C_{(i+1)} + 2 C_i$
- transfert dans le registre opérateur : $10 C_i + C_{(i+1)}$

6 - SP Traitement "EXPRESSION SYMBOLIQUE" (SP6)

Le sous-programme possède deux entrées, la première (AO6) est utilisée à partir de la section "traitement des instructions normales à deux opérandes", la seconde (BO6) par toutes les autres instructions pouvant faire appel à un opérande en mémoire.

Le registre RB (registre branchement) rappelle la nature de l'instruction (RB = - 1 pour un branchement, RB = 0 par l'instruction d'écriture alphanumérique et RB = 1 pour toutes les autres).

Après mise en garde de l'adresse de retour au programme principal et des registres RI, RCI et RTI qui sont utilisés par le sous-programme et ne doivent pas pour autant être détruits, le premier caractère est lu dans ROP ; il indique le signe du premier terme de l'expression et positionne le registre de signe (RSG) à 0 par l'opérateur "-" et à 1 pour

l'opérateur "+" ou l'absence de signe. Le caractère suivant est lu dans (RP).

a) - Orientation du traitement

Si le premier caractère est un chiffre, le terme est une constante décimale et le branchement est effectué au sous-programme correspondant.

Dans le cas contraire :

- le caractère peut être " Δ ", la fin de symbole étant vérifiée (plus, moins ou virgule) le compteur d'adresse est chargé dans le registre opérande et le programme s'oriente vers les test de validité de l'expression.
- le caractère est "X", si le contenu du registre (RP) est "=" le sous-programme "HEXADECIMAL" est appelé, sinon il s'agit d'un symbole (de même que si le 1er caractère est une lettre quelconque).

Si le deuxième caractère n'est pas une séparation (cas d'un symbole d'une seule lettre) le sous-programme de lecture d'un symbole est appelé ; au retour le contenu de (RP) est testé : il doit être "+", "-" ou virgule pour une instruction quelconque. Dans le cas où cela est bien vérifié le sous-programme CADRAGE permet de placer le symbole à gauche dans le mot (B30) quel que soit le nombre des caractères qui le composent.

b) - Recherche du symbole

Si le symbole est placé dans la table, il peut être défini (B18 ou B20 = 0) et l'adresse correspondant au symbole est placée dans le registre opérande (ROP) et la suite de la séquence est exécutée.

Quand le symbole est externe ou non défini, le chaînage des adresses est réalisé en plaçant dans le registre ROP l'adresse affectée au symbole dans la table et en chargeant la valeur du compteur d'adresse (KA) à sa place.

Si le symbole ne figure pas dans la table, après test de la capacité de celle-ci, le pointeur PTRS évolue et le symbole ainsi que son adresse d'utilisation (contenu de KA) sont placés dans la cellule suivante avec l'indicateur "externe" (B20) si l'instruction est quelconque et l'indicateur "non défini" (B18) s'il s'agit d'un branchement.

Après contrôle de la validité de l'expression, le registre opérande est additionné ou soustrait, suivant la valeur de RSG, au registre totalisateur REXP (expression). Si le caractère suivant est une virgule, l'expression est terminée, sinon le registre de signe est positionné et le programme se rebranche à la lecture des deux premiers caractères du terme suivant. (C06).

Le sous-programme est terminé quand l'expression a été jugée valable sur le plan de sa translatabilité.

c) - Validité de l'expression

Quand la zone opérande symbolique comporte un symbole non défini (ou externe), il doit être utilisé seul (cf. chapitre 2 paragraphe VI 3); le contrôle est assuré par le contenu du registre REXT (externe) préalablement remis à zéro. Si le symbole est externe, le registre est diminué de un, quand un symbole défini est rencontré (y compris constantes et symbole d'emplacement actuel " Δ ") la valeur "un" est ajoutée au registre si son contenu est nul à ce moment (il reste sans changement dans le cas contraire).

Ce registre contiendra donc la quantité "1" si l'expression est formée de constantes et symboles définis et "-1" si l'opérande est un symbole externe ou non défini. L'expression sera donc valable si $REXT = \pm 1$ et illégale dans tous les autres cas.

Ce test est évidemment effectué à chaque nouveau terme utilisé dans l'expression, un test global n'apporterait aucun renseignement.

La translatabilité de l'expression s'obtient facilement en ajoutant un pour chaque symbole additionné et retranchant un pour chaque symbole soustrait, au registre RTR préalablement remis à zéro.

Ce registre est utilisé pour générer le code instruction puisqu'il contient un quantité qui dépend de la translatabilité de l'instruction. (0 pour non translatable, 1 pour translatable, -1 pour adresse de chaînage et illégale dans tous les autres cas).

7 - SP "FIN D'INSTRUCTION" (SP7)

Le sous-programme contrôle d'abord le cliché de la fin de l'instruction. Le caractère de séparation est un retour de chariot si l'instruction est terminée et une tabulation si celle-ci est suivie d'un commentaire.

Après passage de la zone de commentaire, si la ligne de programme ne comportait pas d'instruction ((RTR) = 16 ; cas d'une entrée dans le sous-programme par C07) la commande est passée à la séquence qui prépare à l'analyse de l'instruction suivante (W2).

Dans le cas où la capacité de la table des instructions binaires n'est pas dépassée (test sur la valeur du compteur d'adresse) le traitement se poursuit par la transformation du code "translatabilité" en code "instruction" suivant le tableau 18.

(RTR)	Translatabilité	Code instruction	$C_1 = 2 (C_t + 1)$
-1	Chaînage	0	
0	non translatable	2	
1	translatable	4	

— Figure 18 —

Si le contenu du registre RTR est plus grand que un, il ne s'agit pas d'une instruction exécutable et le code ne doit pas être modifié ; les possibilités sont alors les suivantes :

- a) - Constante simple-longueur sans facteur itératif ($(RTR) = 6, (RDL) \geq 0$):
le registre RN (définition de la quantité dont évolue le compteur d'adresse) est positionné à 1 et le registre d'instruction qui contient la constante est chargé en mémoire. (Table binaire).
- b) - Constante double longueur sans facteur itératif ($(RTR) = 6, (RDL) < 0$):
elle est contenue dans les registres RI et ROP1 qui sont alors transférés dans la table des instructions binaires.
- c) - Chargement de constante ou réservation ($(RTR) = 8$) : dans tous les cas, le registre RI est placé en mémoire puisqu'il contient les renseignements nécessaires au chargeur et à l'édition des listes. Ses digits B16 et B17 indiquent en particulier la nature des opérations à effectuer.

Si B16 = B17 = 0, il s'agit d'une réservation de mémoires et RN contient le nombre de places à réserver (évolution de (KA)).

Si B16 ou B17 $\neq$ 0, suivant que la constante avec facteur itératif est en simple ou double longueur, le registre ROP1 ou les deux registres ROP1 et ROP2 sont transférés en table binaire.

Si l'instruction comporte une erreur ($IER \neq 0$), le drapeau d'erreur (digit B1) est placé dans RTR.

La séquence suivante réalise la mise en place du code instruction (primitivement contenu dans le registre RTR) dans la table correspondante. La génération d'une constante en double longueur conduit au chargement de deux codes instruction "constante en simple longueur" consécutifs.

Après évolution du compteur d'adresse, le sous-programme se débranche à l'adresse de retour contenue dans le registre R12.

Nota : pour tout renseignement concernant les problèmes liés à la génération des constantes et à la réservation, consulter le paragraphe VIII 1 b de ce chapitre et le chapitre 4 qui concerne le chargement.

8 - Sous-programmes d'entrée-sortie

Les sous-programmes sont de trois types :

- écriture du contenu d'un mot en caractères alphanumériques
- écriture du contenu d'un mot en caractères hexadécimaux
- lecture de caractères hexadécimaux.

Dans les trois cas, le traitement ne pose pas de problème particulier.

- Pour l'opération d'écriture, l'instruction machine correspondante joue sur les six premiers digits du mot à sortir. Il faut donc amener les caractères successifs, par décalage, dans les positions B1 à B6.
- la lecture, après contrôle de la validité du caractère (0 à 9 et A à F), les caractères sont chargés successivement dans le mot par décalage et addition.

Les programmes correspondants sont donnés en annexe.

ETUDE DU CHARGEUR

Au cours de l'étude détaillée de l'assembleur, nous avons été amenés à définir certaines caractéristiques du chargeur que nous allons maintenant voir en détail.

I - TRAVAIL DU CHARGEUR

Nous avons séparé les tâches du chargeur en deux parties ; l'une destinée à gérer le système complet (assembleur et chargeur), l'autre traitant du chargement proprement dit.

1 - Gestion du système

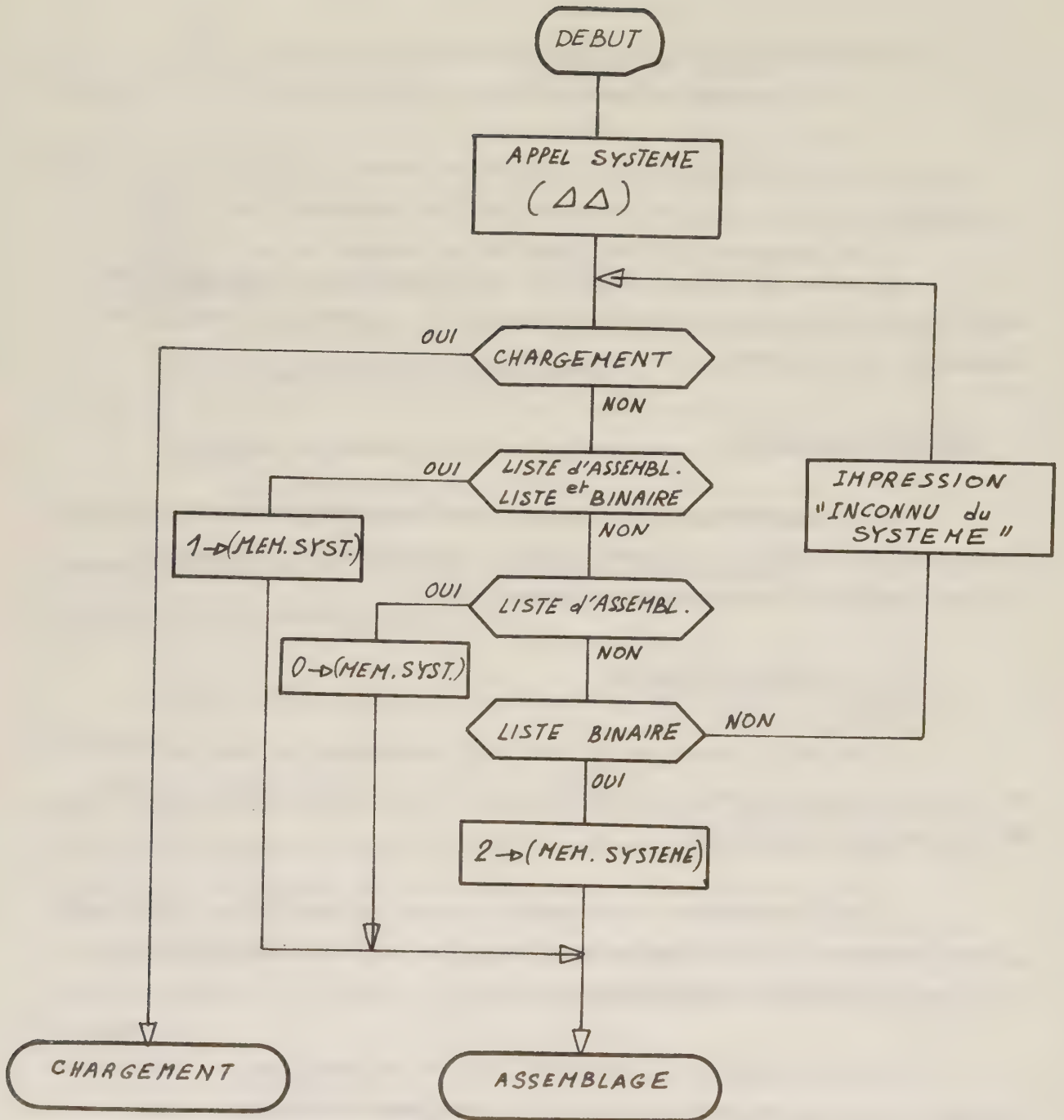
L'ensemble assembleur et chargeur représente un tout du point de vue utilisation puisqu'un programme est normalement assemblé en vue d'un chargement ultérieur.

L'utilisateur doit donc pouvoir par un moyen simple, faire appel soit au chargeur soit à l'assembleur (avec les possibilités d'emploi définies au chapitre 3 paragraphe II - 1).

C'est le rôle de la première section du chargeur qui effectue en quelque sorte, un travail de moniteur en analysant la demande et en orientant le traitement vers telle ou telle partie du système.

2 - Chargement

Nous rappelons que l'assembleur peut fournir une bande binaire comportant les instructions générées et les tables des définitions et références externes.



ORIENTATION du TRAITEMENT.

28 - CHARGEUR P1_ ORG. Fonctionnel

Lorsque le chargeur lit une telle bande il place les instructions en mémoire en effectuant éventuellement la translation des adresses (c'est le code instruction placé devant chaque mot binaire qui indique la nature de celui-ci, cf. Annexe n° 8) et il constitue deux tables à partir des renseignements fournis à la fin de la bande.

Si le programme comporte plusieurs parties, ce n'est qu'après le chargement de toutes les sections que l'assembleur peut effectuer les liaisons entre les différents segments, il contrôle également les cas de multidéfinition et de non définition.

Quand tout ce travail est effectué le système édite une liste de chargement qui indique les différentes sections assemblées et les adresses correspondantes ainsi que la récapitulation des liaisons entre segments.

II - ORIENTATION DU TRAITEMENT

1 - Modification du système existant

Actuellement, lors du départ d'une utilisation quelconque, le décodage des premiers caractères introduits dans la machine conduit à trois types de fonctionnement :

- Langage algorithmique PAF (lecture des trois caractères successifs P.A.F.)
- Code spécial dit "S" (le 1er caractère est un espace arrière)
- Ordres initiaux (langage autocode) dans tous les autres cas.

Aucune modification à ce niveau n'est possible, la séquence de décodage ne pouvant être modifiée par programmation. C'est donc au début des ordres initiaux que nous placerons le test d'orientation vers le système ajouté.

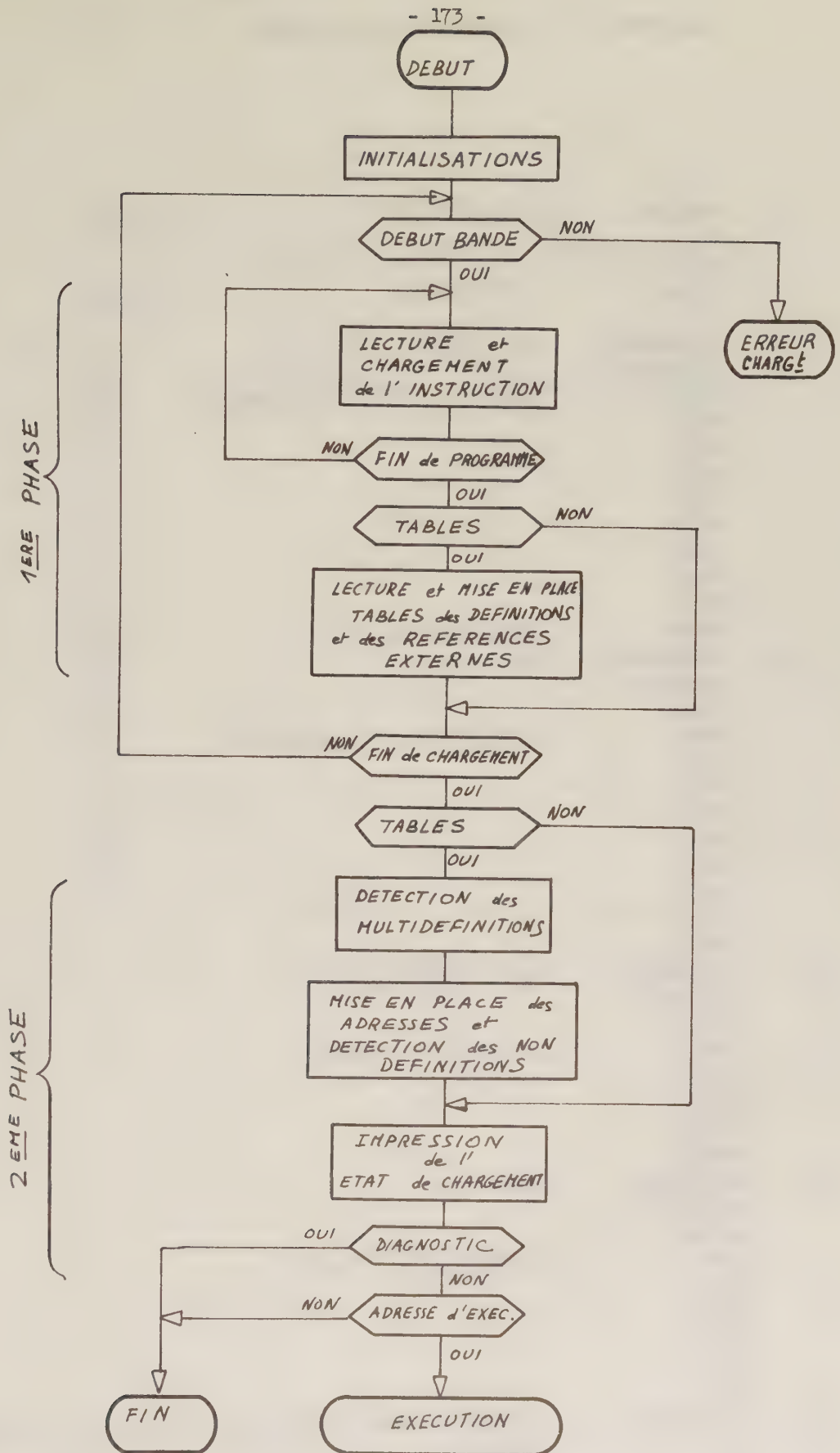
L'appel au moniteur sera effectué conventionnellement par la suite des caractères : $\Delta \Delta$ qui ne correspond à rien du point de vue des ordres initiaux.

2 - Le chargeur (partie 1) ayant été appelé, une série de tests en chaîne est réalisée sur les éventualités ci-après ; les deux premières lettres sont lues et s'il y a reconnaissance, le système complète l'écriture du mot :

- $\Delta \Delta$ CHARGEMENT, orientation du traitement sur le chargement de programmes (chargeur P2).
- $\Delta \Delta$ ASSEMBLAGE, assemblage et édition de la liste d'assemblage et de la bande binaire objet.
- $\Delta \Delta$ LISTE, assemblage et impression de la liste correspondante.
- $\Delta \Delta$ BINAIRE, assemblage et perforation de la bande binaire objet.

Si les deux caractères qui suivent l'appel ($\Delta \Delta$), ne correspondent à aucune des possibilités ci-dessus, le diagnostic "INCONNU DU SYSTEME" est imprimé et le programme se rebranche au début de la séquence d'appel.

Dans le cas où un assemblage est demandé, le système place dans une mémoire dont il connaît l'adresse, une valeur conventionnelle (0, 1 ou 2) qui permet lors de l'édition des listes, de savoir quel est le travail réel à effectuer (cf. chapitre 3 VIII) ; la table des codes instruction est remise à zéro (cette opération est nécessaire au fonctionnement correct de l'assembleur).



CHARGEUR P1 : ORIENTATION DU SYSTEME

0	AOELOS	LECTURE C1
	X	
	KOTV6	
	A13LOS	LECTURE C2
	X	
	V13QV2280	TEST "CH"
	RV17AY14	
	EV33	IMPRESSION : "ARGEMENT"
	EV50	
	EV39	
	EV37	
	EV45	
	EV37	
	EV46	
	EV52	
	EV25	
	RV80A	A (CHARGEUR P2)
17	V13QV2063	TEST "AS"
	RV29AY14	
	A1EV0	
	EV51	IMPRESSION "SEMBLAGE"
	EV37	
	EV45	
	EV34	
	EV44	
	EV33	
	EV39	
	EV37	
	RV45A	
29	V13QV2857	TEST "LI"
	RV37AY14	
	A1EV1	
	EV51	IMPRESSION "STE"
	EV52	
	EV37	
	EV25	
	RV46A	
37	V13QV2217	TEST "BI"
	RV57AY14	
	A1EV2	
	EV46	IMPRESSION "NAIRE"
	EV33	
	EV41	
	EV50	
	EV37	
45	EV25	
46	A15EM47AS	REMISE A ZERO TABLE DES CODES INSTRUCTION
47	X	A (TABCI)
	AOEVO	
49	TCMO+	

	A15V1	
	V15TM52ΔS	
52	X	DERNIERE ADRESSE (TABCI)
	RV49ΔY13	
	T1EM55ΔS	
55	X	MEMOIRE SYSTEME
	RM79Δ	VERS ASSEMBLAGE
57	EV25	IMPRESSION "INCONNU DU SYSTEME"
	EV41	
	EV46	
	EV35	
	EV47	
	EV46	
	EV46	
	EV53	
	EV15	
	EV36	
	EV53	
	EV15	
	EV51	
	EV57	
	EV51	
	EV52	
	EV37	
	EV45	
	EV37	
	EV15	
	RV(Δ	RETOUR AU DEBUT DU PROGRAMME
	F	
79	X	A (ASSEMBLAGE)
	FE	FIN DE BANDE

3 - Organigrammes et programmes

Cette partie ne posant aucun problème particulier, nous n'expliquons pas davantage la réalisation du programme correspondant.

III - CHARGEMENT DES PROGRAMMES

1 - Tables utilisées

Au cours du chargement, trois tables sont construites :

- a) - La table des programmes indique l'adresse d'exécution du programme complet, le nom symbolique (éventuellement) et les adresses de début et de fin de chaque segment chargé en mémoire.

Un compteur KP donne le nombre de sections chargées et permet d'obtenir, par un calcul simple, l'adresse de la première cellule libre de la table (figure 19).

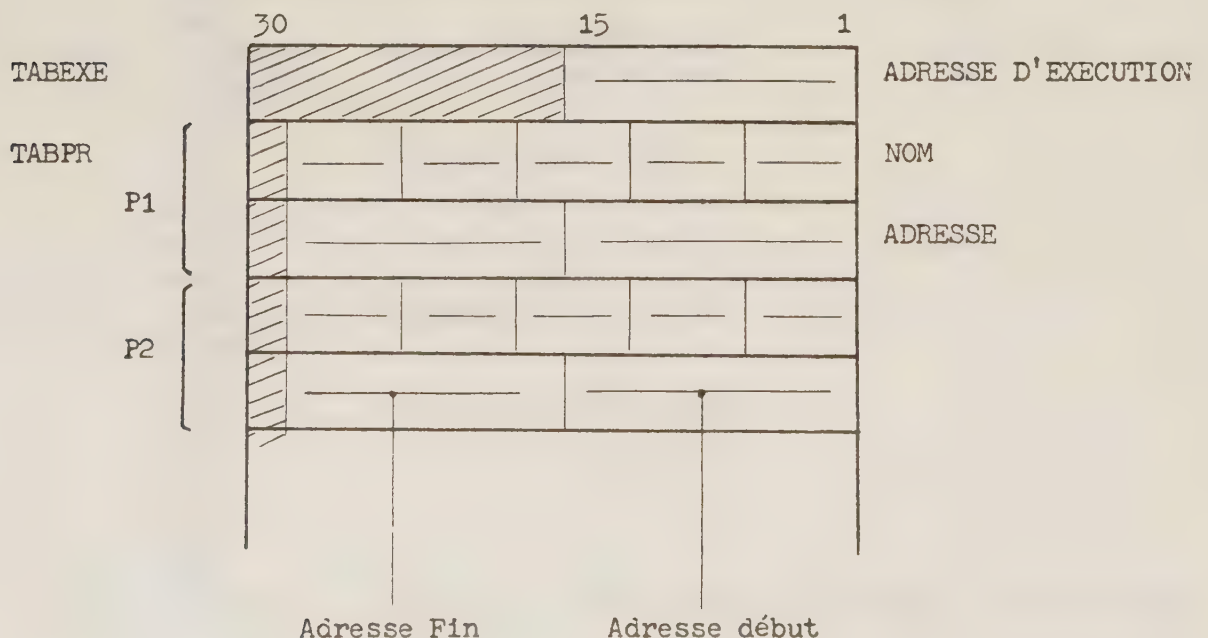


Figure 19

- b) - Tables des définitions et références externes (figure 20a): chaque segment assemblé est généralement complété par des définitions et références externes qui permettent de réaliser les liaisons. Pour chaque nouvelle section à charger, ces renseignements sont stockés dans leurs tables respectives.

Chacune des tables est constituée de la même façon ; un pointeur fournit l'adresse de la première cellule libre. Une méthode d'enchaînement permet de faciliter l'utilisation ultérieure de ces listes (2ème phase du chargement) : chaque sous-tableau est précédé d'une position de mémoire qui indique le numéro du segment auquel il appartient et l'adresse de la section suivante. La reconnaissance de ces mots de contrôle est obtenue par la valeur "1" des digits 32 et 33 ($B_{32} = B_{33} = 0$ pour toute cellule normale).

2 - Chargement 1ère phase : (organigrammes 1-2-3-4)

Après initialisation de RAD (registre d'adresse de chargement du segment de programme), PTR2 et PTR3 (registres pointeurs des tables des définitions et références externes) et remise à zéro de REXE (registre indiquant si l'adresse d'exécution du programme est définie convenablement) ADEXE (adresse d'exécution) et KP (compteur de programme), la lecture de la bande commence par "l'ignorance" des codes avance-bande (32 en code interne) et la reconnaissance du premier caractère de contrôle (B_{16}) qui indique que les quatre caractères suivants représentent la taille du programme et les huit autres son nom symbolique, à partir de ces renseignements la table des programmes est mise à jour : RAD est transféré dans le registre de base (RB) avant de contenir l'adresse de chargement de la prochaine section. L'incrémentation du compteur de programme est suivie de l'initialisation du registre d'index à la valeur du registre de base qui contient l'adresse de chargement.

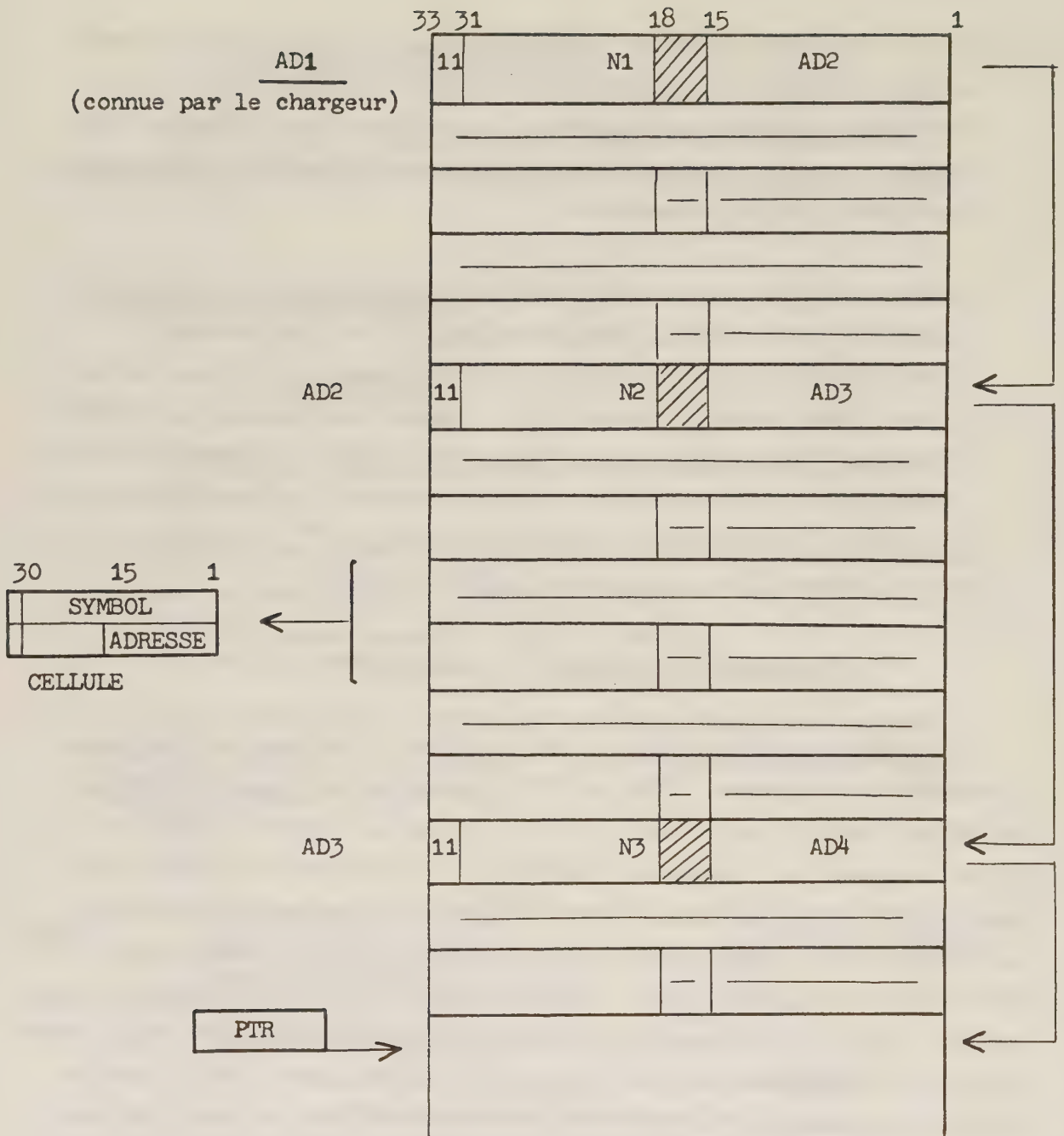


Figure 20a - Chargement - Table "externes"

La mise en place des mots successifs s'effectue de la façon suivante : un premier test sur les digits 4 et 6 du caractère de contrôle (code instruction) indique s'il s'agit d'un code spécial (constantes multiples et fin de programme).

- a) - Dans le cas contraire, un test sur les digits 1 et 2 permet de déterminer si le mot est une constante ou une instruction non translatable. Si l'instruction est translatable, le registre de base préalablement cadré (en B12 pour les branchements et en B5 pour les ordres quelconques) est additionné à l'instruction avant transfert en mémoire.

Les adresses de chaînages (RT = 0) sont évidemment translatées sauf si elles sont initialement nulles car il s'agit alors de la dernière adresse d'une chaîne qui doit rester nulle pour être détectée lors de la mise en place ultérieures des adresses correspondant aux références externes, après chargement en mémoire, l'index est incrémenté et la séquence recommence.

- b) - Le code instruction est un code "spécial" si (RT) = 8, il s'agit d'une constante multiple ou d'une réservation ; les digits B16 et B17 donnent la nature des opérations à effectuer.

- B16 = B17 = 0 représente une réservation de mémoires et la valeur du mot binaire lu dans RI (registre instruction) sur 8 caractères hexadécimaux est ajoutée au registre d'index.
- B16 = 1, B17 = 0 signifie que le mot suivant (8 caractères) est une constante simple longueur ; le contenu de (RI) fixe le nombre de répétitions à effectuer.
- B16 = 0, B17 = 1 indique que les deux mots suivants représentent une constante en double longueur et que RI donne encore le facteur itératif.

Après chargement de la constante dans les conditions indiquées, on passe à l'instruction suivante.

Si (RT) $\neq$ 8, il ne peut s'agir que du code fin de programme et le mot qui suit fournit éventuellement l'adresse d'exécution (si le contenu du mot est négatif, il n'y a pas d'adresse d'exécution dans cette section). Le digit B18 permet de savoir si l'adresse de début est translatable (cf. directive "FIN") dans ce cas, le contenu du registre de base lui est ajouté et elle est transférée en tête de la table des programmes (ADEXE) et la valeur du registre REXE est augmentée de 1.

- c) - Quand la fin du programme proprement dit a été détectée, le code de contrôle suivant doit indiquer le début de la table des définitions externes (E16) dont la taille est fournie par la lecture de deux caractères. Si la table existe (RI $\neq$ 0) elle est chargée en mémoire par appel à un sous-programme dont le rôle est de générer le mot de contrôle correspondant au sous-tableau pour lequel il faut lire les symboles (8 caractères) et les adresses (4 caractères). La table étant chargée, le caractère de contrôle (F16) doit ensuite précéder la table des références externes qui est traitée de la même façon. Après mise en place des tables, le système ignore les codes de fin de bande et teste le premier caractère différent : si celui-ci indique la fin du chargement (D16), cela signifie qu'il n'y a pas d'autre section à charger et l'on passe alors à la deuxième phase du chargement qui consiste à réaliser les liaisons entre segments.

Dans le cas contraire, le programme s'oriente sur le début de la séquence pour la lecture d'une nouvelle bande binaire.

3 - Chargement 2ème phase (organigrammes 5-6-7-8)

Il comporte en fait deux opérations enchevêtrées, l'impression de l'état de chargement et la réalisation des liaisons externes. Les opérations de contrôle et l'édition sont faites simultanément afin de réduire le temps d'exécution en minimisant le nombre d'appels à la mémoire centrale.

Les quatre sections secondaires de cette phase sont :

- impression de la table des programmes chargés,
- contrôle et impression des définitions externes, détection des multi-définitions,
- contrôle et impression des références externes, mise en place des adresses dans les instructions incomplètes, détection des cas de non définition,
- contrôle de possibilité et exécution éventuelle du programme.

a) - Toutes les sections étant chargées, il est alors possible de déclencher l'impression de la table des programmes placés en mémoire. Cette liste de chargement permet de connaître les adresses réelles des symboles pour vérifications ultérieures et de se rendre compte de l'implantation des programmes en mémoire.

Après impression du titre :

"ETAT DE CHARGEMENT PAR CAB 502 B OCT 63"

et de la date de chargement, cette table fournit pour chaque segment : le numéro du programme (NO PR), le nom symbolique (NOM), l'adresse de chargement (DEBUT) et la dernière adresse du programme (FIN). Le registre d'index préalablement chargé fournit l'adresse courante de la cellule de la table. L'impression des 5 caractères contenus dans la première mémoire donne le nom du programme (5 blancs éventuellement) tandis que sous forme hexadécimale la sortie des digits 1 à 15 et 16 à 30 du mot suivant correspond aux limites du programme.

L'accumulateur logique ACL1 contient le numéro de chargement du programme, la comparaison avec KP (compteur de programme), permet de savoir si la table est terminée. Dans le cas contraire, l'index est incrémenté de 2 et le programme se rebranche au début de la séquence.

- b) - La phase suivante consiste à comparer les éléments de la table des définitions externes pour détecter les multidéfinitions éventuelles. Comme nous l'avons déjà fait remarquer, l'impression des symboles et de leurs adresses réelles est effectuée ici ; pour les tests, ces mots sont transférés de la mémoire centrale (tambour) dans les registres généraux et cela évite d'avoir à les appeler par la suite.

L'intérêt de l'organisation d'une telle table en sous-tableau est double :

- la présentation la plus logique de la liste des symboles externes est celle qui regroupent ceux-ci par section de programme facilitant ainsi en cas d'erreur, les contrôles du programmeur. Le traitement correspondant devient alors très simple puisque les sous-tableaux sont fournis tels quels sur la bande d'assemblage.
- la comparaison d'un symbole n'a pas besoin d'être faite par rapport à toute la table puisqu'a priori il n'y a pas de multidéfinition au niveau d'un tableau élémentaire, cette éventualité ayant été testée et diagnostiquée au cours de l'assemblage (cf. chapitre 3 paragraphe V - 1). Il suffit donc de comparer chaque élément d'un sous-tableau à chaque symbole contenu dans les tables secondaires suivantes ; le nombre d'opérations est alors plus faible que pour une vérification (ou un classement alphabétique) dans un tableau unique.

L'impression en cours de traitement fournit par segment chargé, les symboles (définitions) externes et les adresses absolues respectives. Si le symbole considéré est multiplement défini (une ou plusieurs sections), le diagnostic *MULTIDEF.* est édité vis à vis de la première définition ainsi que la suite des adresses où le symbole est défini ; pour les fois suivantes, le système indique seulement "MULTIDEF", il suffit de se reporter au premier symbole de la chaîne pour connaître toutes les adresses de définition.

L'index ayant été préalablement chargé par l'adresse de tête du tableau DEFEX, la comparaison au pointeur de la table (PTR2) indique si celle-ci contient des éléments ; dans le cas contraire, un registre de test (REXT) est positionné à 1 ((REXT) = 0 si la table n'est pas vide) et le programme passe à la phase suivante (G.traitement des références externes).

Le titre "DEFINITIONS EXTERNES" est édité ainsi que le sous-titre "PROGRAMME N", N étant le numéro de la section chargée. Le pointeur PTR1 caractérise le 1er élément de la comparaison à effectuer, le registre RN contient l'adresse du tableau suivant, RDEF est un registre dont le contenu est nul si le 1er terme de la comparaison est uniquement défini et différent de zéro dans le cas de multidéfinition.

Symbole et partie adresse du 1er élément sont successivement placés en registres (RT1 et REC) et imprimés.

Le digit B20 de la zone d'adresse est conventionnellement choisi pour reprérer si un élément de la table a déjà été reconnu comme multi-défini (B20 = 1) par rapport aux sous-tableaux qui le précèdent ; dans ce cas, le symbole considéré est imprimé ainsi que le diagnostic "MULTIDEF" et la cellule suivante est appelée par incrémentation de deux positions de l'index, pour un nouveau cycle de comparaisons. Il n'est pas utile d'exécuter les tests sur un tel symbole puisque tous les renseignements

concernant cette multidéfinition ont été obtenus et imprimés lors des opérations faites sur le 1er symbole de la chaîne. Quand le traitement arrive au dernier tableau (égalité des registres RN et PTR2) chaque définition est imprimée, tous les éléments du sous-tableau étant décrits par évolution du pointeur PTR1 ; la séquence est achevée quand l'égalité entre PTR1 et RN est obtenue.

Dans le cas général, le 2ème élément de la comparaison (à charger dans le registre de test RT2) s'obtient en plaçant en index la valeur du registre RN plus un ; il suffit ensuite de comparer les deux symboles.

S'ils sont différents le deuxième terme qui suit est appelé par incrémentation de deux de l'index, la fin de boucle est détectée quand (RN) est égal à (PTR2). Le passage d'un sous-tableau à un autre (évolution d'index d'une unité) est détecté par le test du signe de RT2 car dans ce cas il contient un mot de chaînage ($B33 = B32 = 1 \longleftrightarrow$ contenu négatif). Si les deux symboles à comparer sont égaux, il y a multidéfinition et le diagnostic "***MULTIDEF***" est imprimé pour la première définition de la chaîne (RDEF = 0), après évolution de un du registre de définition, la partie adresse du 2ème élément de comparaison est éditée et reçoit le digit B20 ; le compteur de diagnostic DG et l'index évoluent respectivement de une et de deux unités. Le test de (RX) par rapport à PTR2 permet de déterminer la fin du cycle de comparaisons relatif à un élément. L'appel du prochain symbole à examiner est fait après incrémentation de deux du pointeur PTR1. Tant que celui-ci est inférieur à RN, le programme se rebranche à l'impression du symbole et de son adresse ; quand (PTR1) devient égal à (RN) cela signifie que le sous-tableau est terminé. Si les registres PTR2, RN et PTR1 contiennent la même adresse le traitement des définitions externes est terminé (tous les éléments de la table ont été traités) et la phase suivante est exécutée ; dans le cas contraire, le programme est buclé en F, séquence qui correspond aux opérations décrites ci-dessus (impression du sous titre, impression des symboles, comparaisons, ...).

Remarque : La fig. 20b donne les états respectifs des registres PTR1, PTR2, RN, RX dans les phases particulières du traitement.

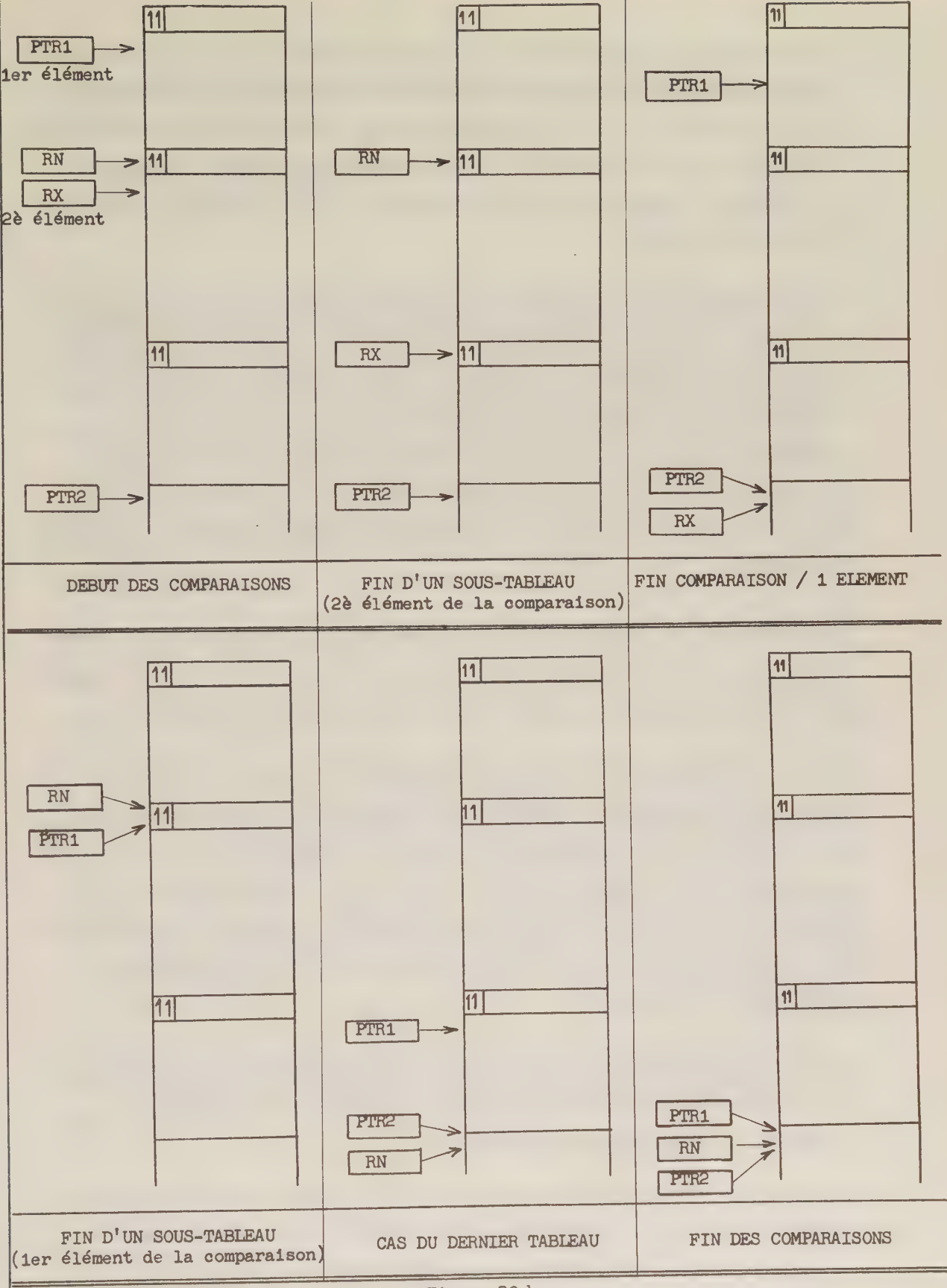


Figure 20 b

- c) - Le traitement des références externes est mené d'une façon similaire à celui des définitions externes en ce qui concerne la gestion des sous-tableaux et de leur impression. Mais cette fois les deux tables sont utilisées simultanément, les comparaisons faisant intervenir pour premier terme une référence externe et pour deuxième terme une définition externe.

Chaque référence externe est comparée séquentiellement aux symboles du tableau DEFEX ; si l'identité apparaît au cours de la recherche, l'adresse de définition est placée dans la chaîne des instructions à compléter (la dernière adresse de la chaîne est celle de la référence externe, la méthode de liste enchaînée est identique à celle utilisée au cours de l'assemblage). S'il n'existe pas de définition correspondant à la référence externe, celle-ci est suivie du diagnostic "*NON DEF.*".

La recherche par balayage étant stoppée dès la reconnaissance d'une identité, si la définition externe est du type multidéfini, l'adresse affectée à la référence sera celle de la première définition.

L'index, contenant au départ du traitement l'adresse de tête du tableau des références externes, est comparé au pointeur correspondant (PTR3). S'il y a égalité la table est vide ; dans le cas où il existe des définitions externes (REXT = 0) la machine imprime le message "DEFINITIONS EXTERNES NON UTILISEES", ceci est une indication pour le programmeur mais n'influe pas sur la mise en exécution ultérieure du programme.

Si la table des définitions externes est également vide (REXT = 1) la commande est directement passée à la phase de contrôle et d'exécution du programme (repère I).

Comme précédemment, la mise en page du tableau et du premier sous-tableau est éditée :

REFERENCES EXTERNES

PROGRAMME 1

Le pointeur PTR1 caractérise le premier symbole de la comparaison (référence externe), le registre RADE contient l'adresse de tête du premier sous-tableau de la table DEFEX (définitions externes) et RN représente l'adresse du tableau secondaire suivant la référence externe en cours de traitement. Après impression du symbole et de l'adresse de la référence externe, si le tableau DEFEX examiné auparavant est vide, ($REXT \neq 0$), on passe directement à l'impression du diagnostic de non définition et à l'évolution du compteur correspondant (DG). Dans le cas contraire, ($REXT = 0$), un appel indexé ($RX = RADE$ au début de la boucle) fournit la définition externe qui est le 2ème élément de la comparaison ; si un sous-tableau se termine ($RT2 = 0$) une position de mémoire est sautée et l'on retourne à la séquence d'appel à moins que la table DEFEX soit terminée ($RX = PTR2$) ce qui signifie que la référence externe n'est pas définie. Dans le cas général ($RT2 \geq 0$) si les deux symboles sont différents ($RT2 \neq RT1$) l'index est augmenté de 2 et le programme se boucle sur l'appel de la définition externe suivante sauf si la table est terminée (cas de non définition) ; si les symboles sont identiques ($RT1 = RT2$), l'appel de la mémoire suivante de la table DEFEX fournit l'adresse de définition de la référence externe qui est ensuite imprimée sur l'état de chargement et, la mise en place de cette adresse aux endroits voulus peut être exécutée (la première instruction à pourvoir étant celle qui est associée à la référence externe dans le tableau REFEX).

L'incrémentation de deux unités du pointeur PTR1 permet d'atteindre la référence externe suivante et de retourner à la phase d'impression si le sous-tableau en cours n'est pas achevé ($PTR1 < RN$) ; dans

le cas où (PTR1) = (RN) un sous-tableau vient d'être terminé, si (RN) est alors égal à PTR3 (pointeur de la table REFEX) le travail concernant les références externes est achevé et le programme passe à la phase finale (I), sinon il traite un nouveau sous-tableau en se bouclant au début de la phase d'impression du sous-titre.

- d) - Les liaisons entre les segments du programme ayant été réalisées, celui-ci peut être exécuté. Toutefois, afin d'éviter d'effectuer un traitement conduisant à des erreurs plus ou moins graves, la commande n'est donnée qu'après certains contrôles.

Le système imprime tout d'abord le contenu du compteur DG qui indique le nombre de diagnostics émis au cours de la deuxième phase du chargement (multidéfinitions et non définitions). Si ce nombre n'est pas nul, le programme n'est pas exécuté et le message "NON EXECUTION" apparaît en regard.

Dans le cas contraire, le registre REXE positionné au cours de la 1ère phase de chargement, est testé ; si son contenu est nul, cela signifie qu'aucune des sections chargées ne comportait d'adresse d'exécution du programme, celui-ci ne sera donc pas exécuté et le message correspondant "PAS D'ADRESSE D'EXECUTION" est édité.

Un dernier test est encore fait sur le contenu du registre REXE. Si plusieurs segments définissent chacun une adresse d'exécution (REXE) est différent de 1, le système refuse l'exécution et termine sur l'impression du diagnostic :

"PLUSIEURS ADRESSES D'EXECUTION"

Le contenu de REXE étant égal à 1, il existe une adresse d'exécution et une seule pour définir le point de départ du programme à exécuter ;

après édition du message :

"ADRESSE D'EXECUTION N PR M "

le contrôle est inconditionnellement passé à cette adresse qui est logée dans la première mémoire de la table programme (cf. chargement des programmes paragraphe III. 1).

Nota : des exemples de chargement de programmes "test" décrivant les diverses possibilités peuvent être consultés en annexe.

4 - Organigrammes et programmes

Nous n'insistons pas davantage sur l'algorithme du traitement décrit en détail dans les pages précédentes.

Les sous-programmes de lecture et d'écriture de caractères hexadécimaux et DCB sont ceux qui ont déjà été utilisés dans l'assemblage. La lecture des tables de définitions et références externes fournies sur le ruban d'assemblage est effectuée à partir d'un sous-programme dont le travail s'effectue en deux temps.

a) - Initialisations et création du mot de contrôle : la première adresse libre du tableau (adresse de tête lors de l'enregistrement de la première table) est transmise au registre d'index par l'intermédiaire de l'accumulateur logique 1 (ACL1) qui joue le rôle de pointeur (il est chargé, avant l'entrée dans le sous-programme, par le pointeur de la table correspondante) ; le mot de contrôle est formé dans le registre accumulateur logique 2 (ACL2).

Le compteur KP fournit le numéro d'ordre de la section chargée ; après cadrage en position 18 du registre ACL2. les contenus de l'index et de RI (registre contenant la taille du sous-tableau) plus un , y sont ajoutés pour obtenir l'adresse du sous-tableau suivant. Le mot de contrôle est alors complété par les digits de

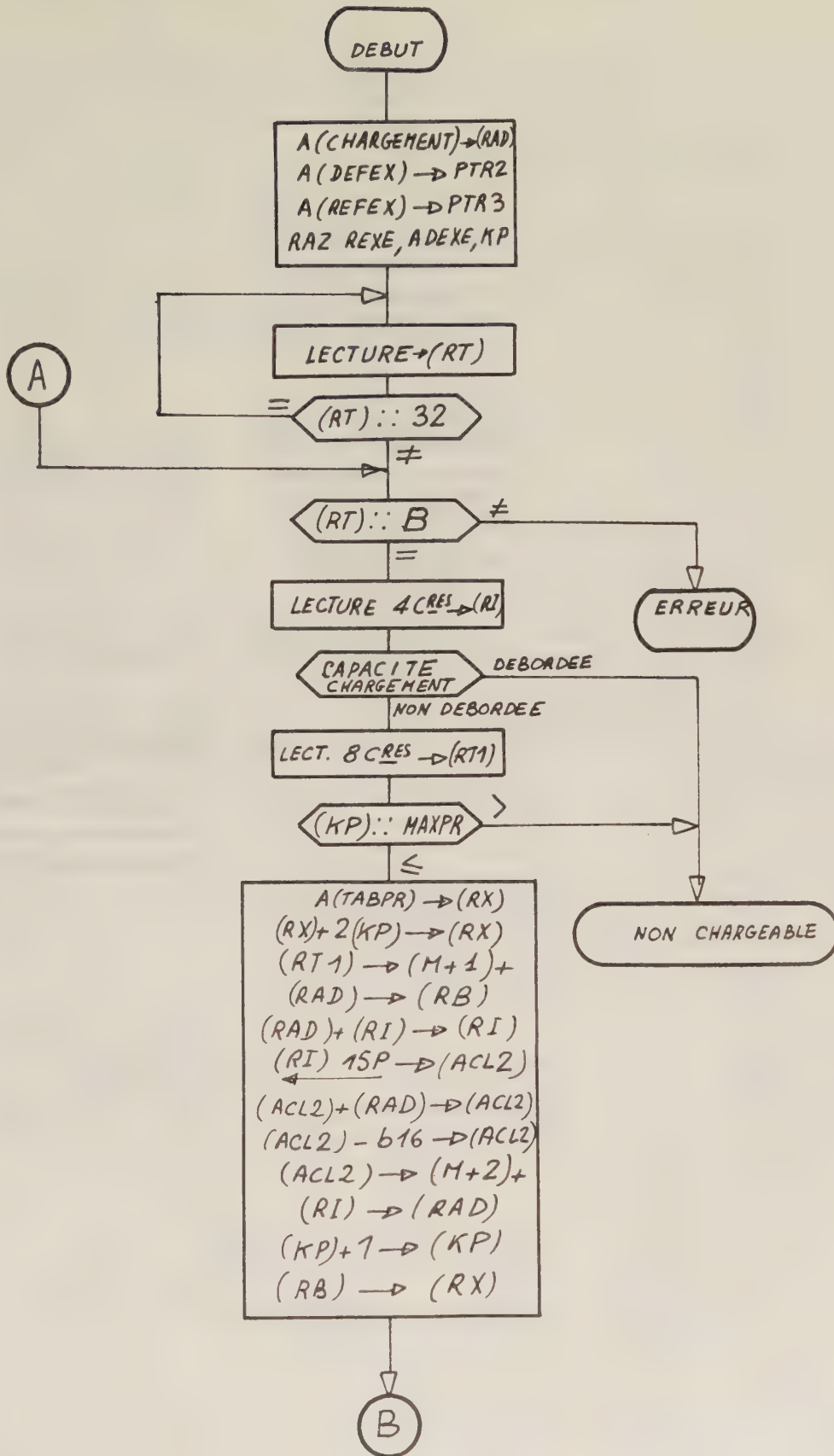
reconnaissance ($B33 = B32 = 1$) avant transfert indexé dans la table ; cette opération est suivie de l'incrémentation d'une unité du registre ACL1 pour la mise à jour du pointage du tableau en cours de lecture.

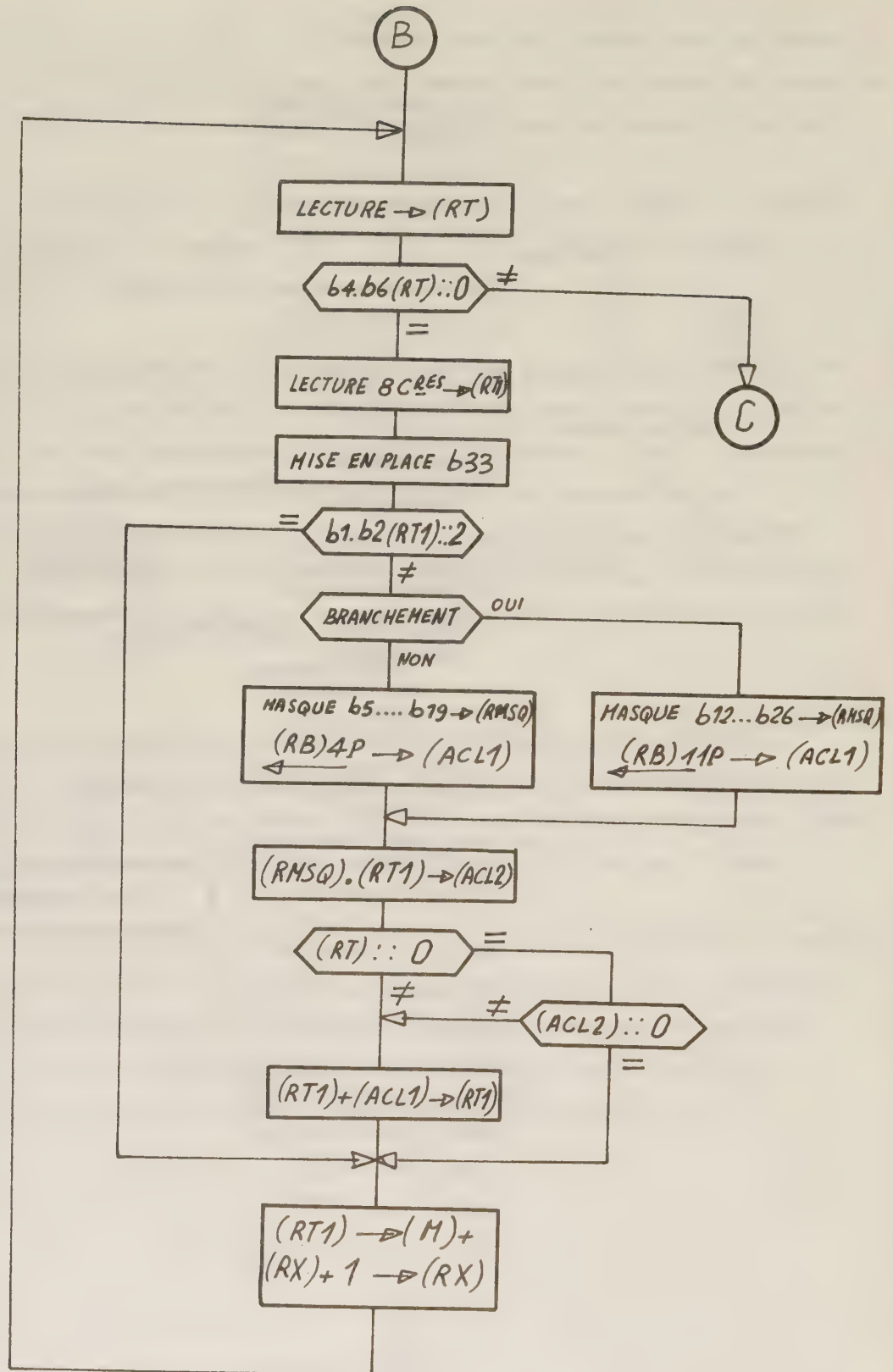
Le registre RATM qui contient au départ l'adresse de tête du tableau est augmenté de la valeur "384" pour définir l'adresse maximum permise. (cf. Remarque).

- b) - Lecture et chargement de la table : l'index étant initialisé par le pointeur, la lecture de 8 caractères hexadécimaux fournit la partie symbole qui est directement transférée en mémoire ; les 4 caractères suivants constituent la partie adresse qui est translatée par addition du registre RB qui contient l'adresse de base du segment chargé. Après mise en place dans la mémoire suivante, ACL1 évolue de deux unités et est comparé à (RATM) pour tester le débordement de la table.

Dans le cas contraire, le registre RI contenant initialement la taille du sous-tableau est diminué de 2 ; quand son contenu est nul la table est totalement chargée et l'on sort du sous-programme, sinon la séquence se rebranche à une nouvelle modification de l'index.

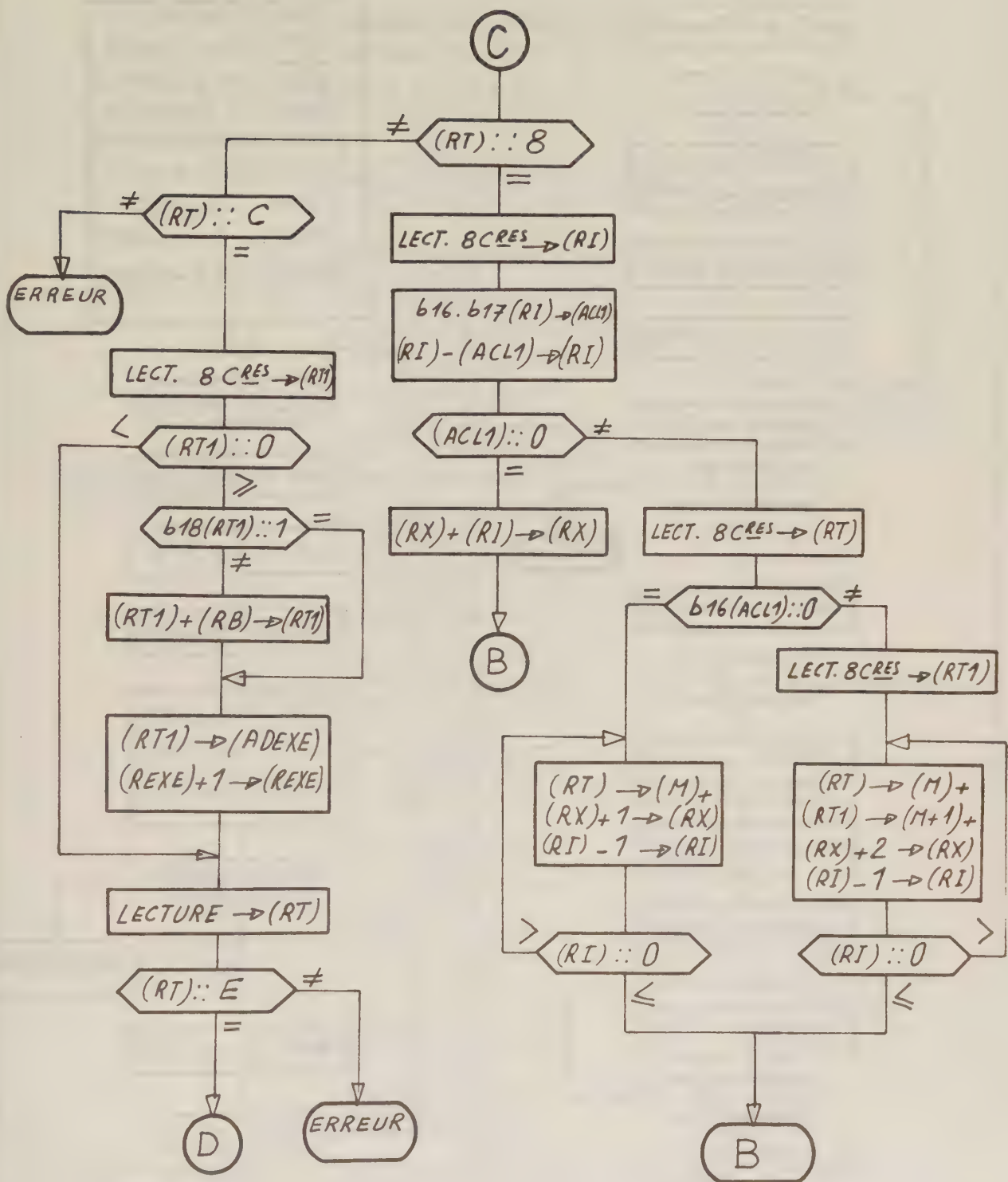
Remarque : Le nombre 384 qui donne la taille maximale des tables est choisi parce qu'il est multiple de 128 et représente ainsi un nombre entier de pistes du tambour constituant la mémoire centrale. Le fait de modifier cette quantité ne poserait d'ailleurs aucun problème, dans la mesure de la place disponible.

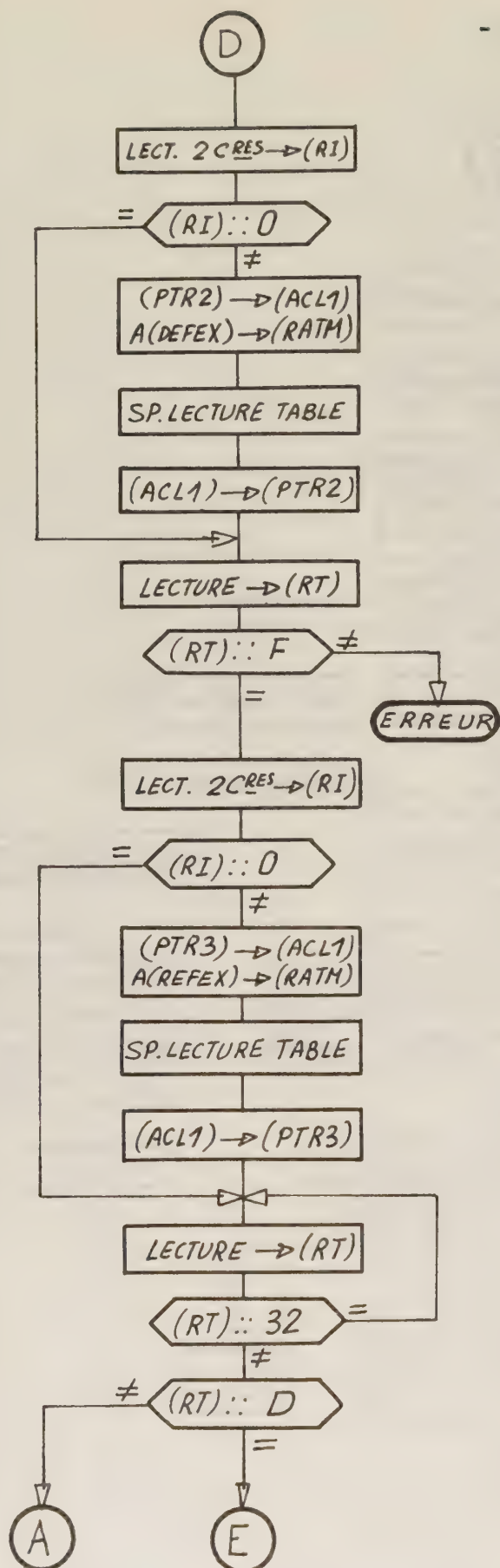




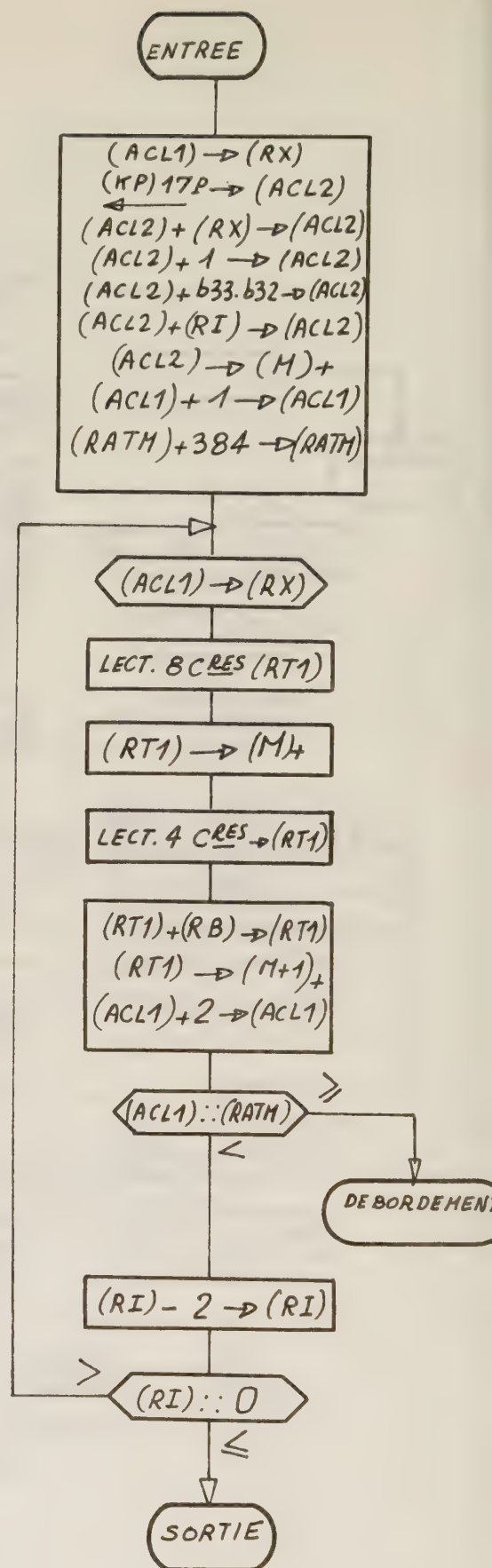
INSTRUCTIONS et CONSTANTES

31_CHARGEUR P2 - ORG. Machine (2)



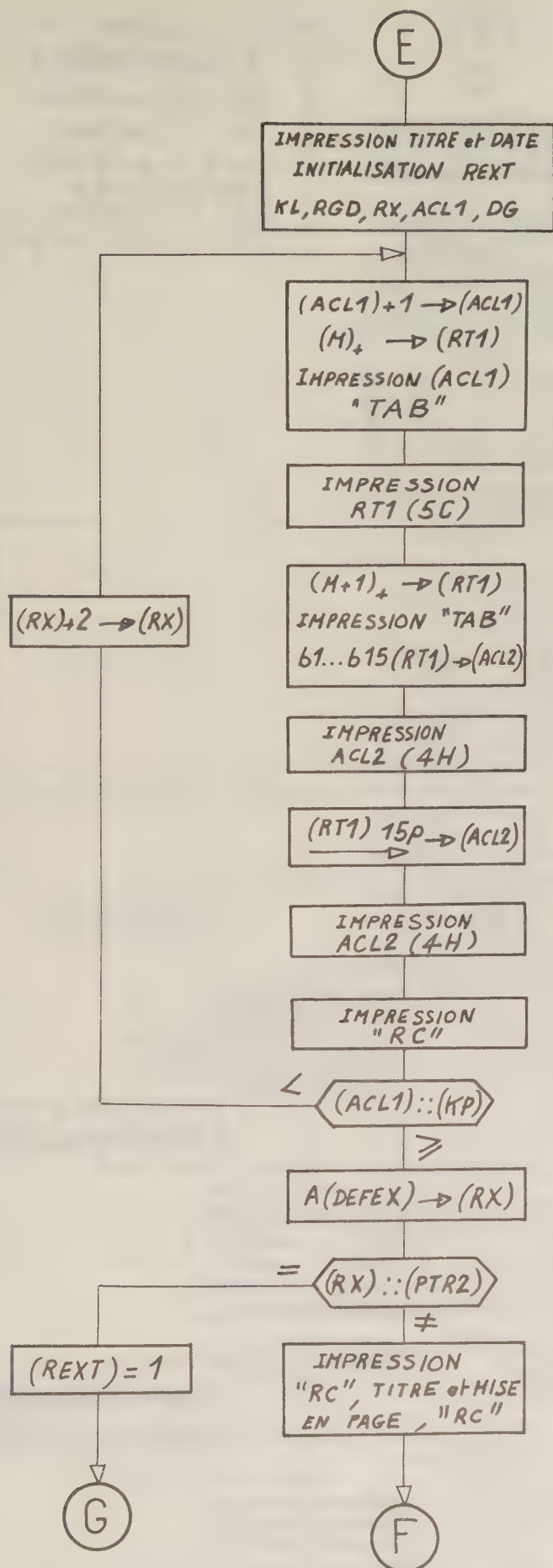


LECTURE des TABLES

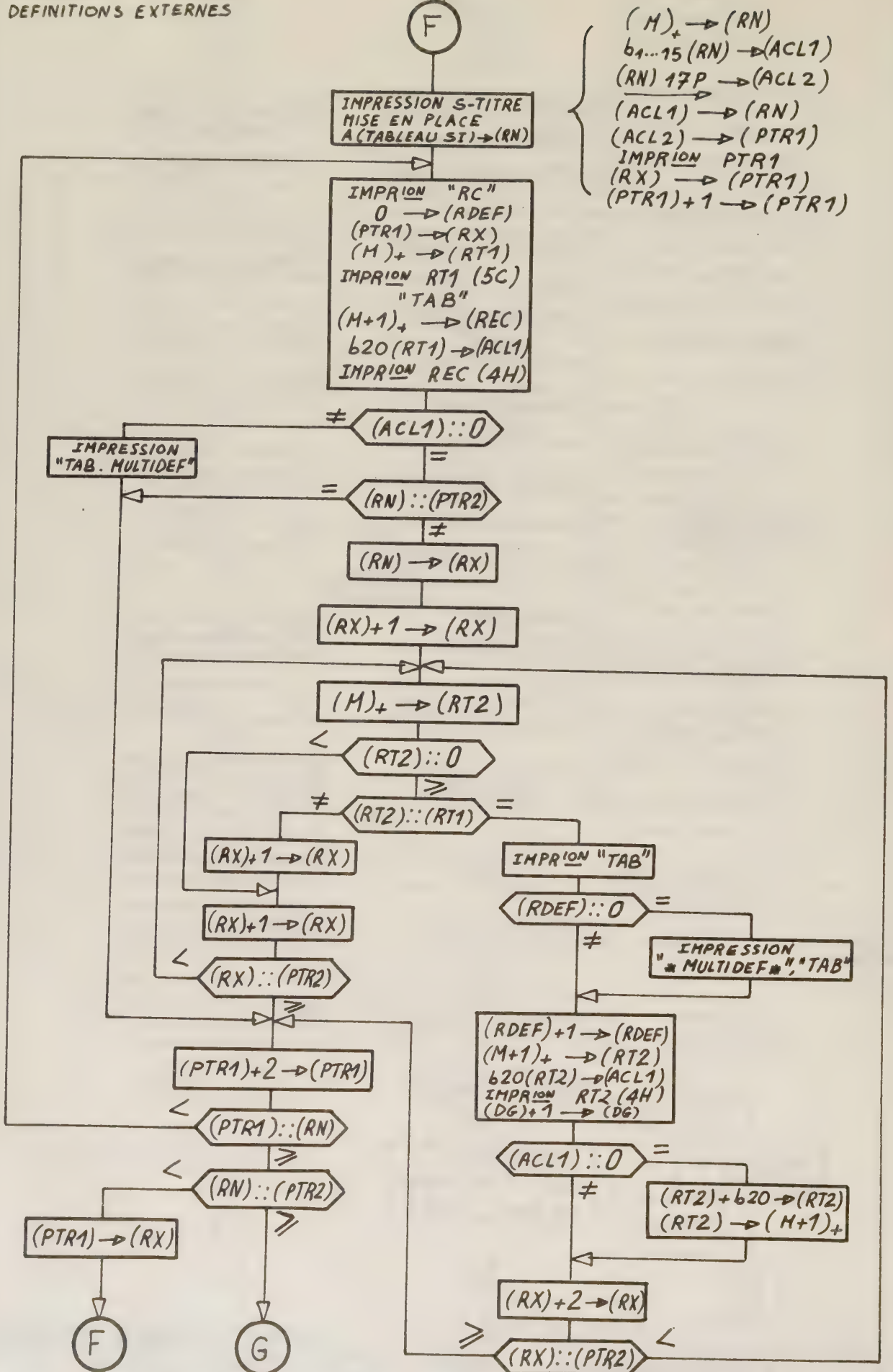


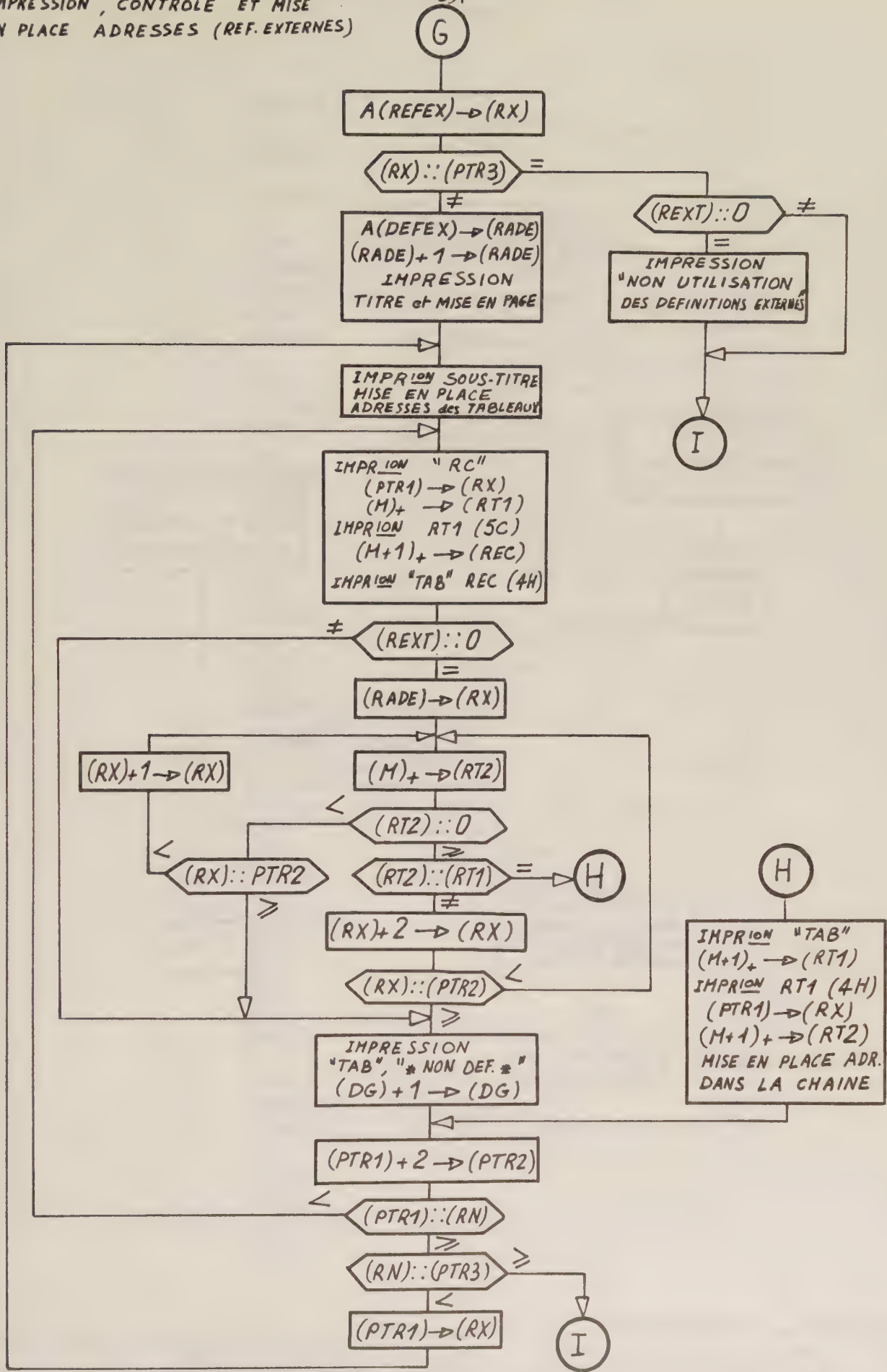
SP. LECTURE TABLE (SP15)

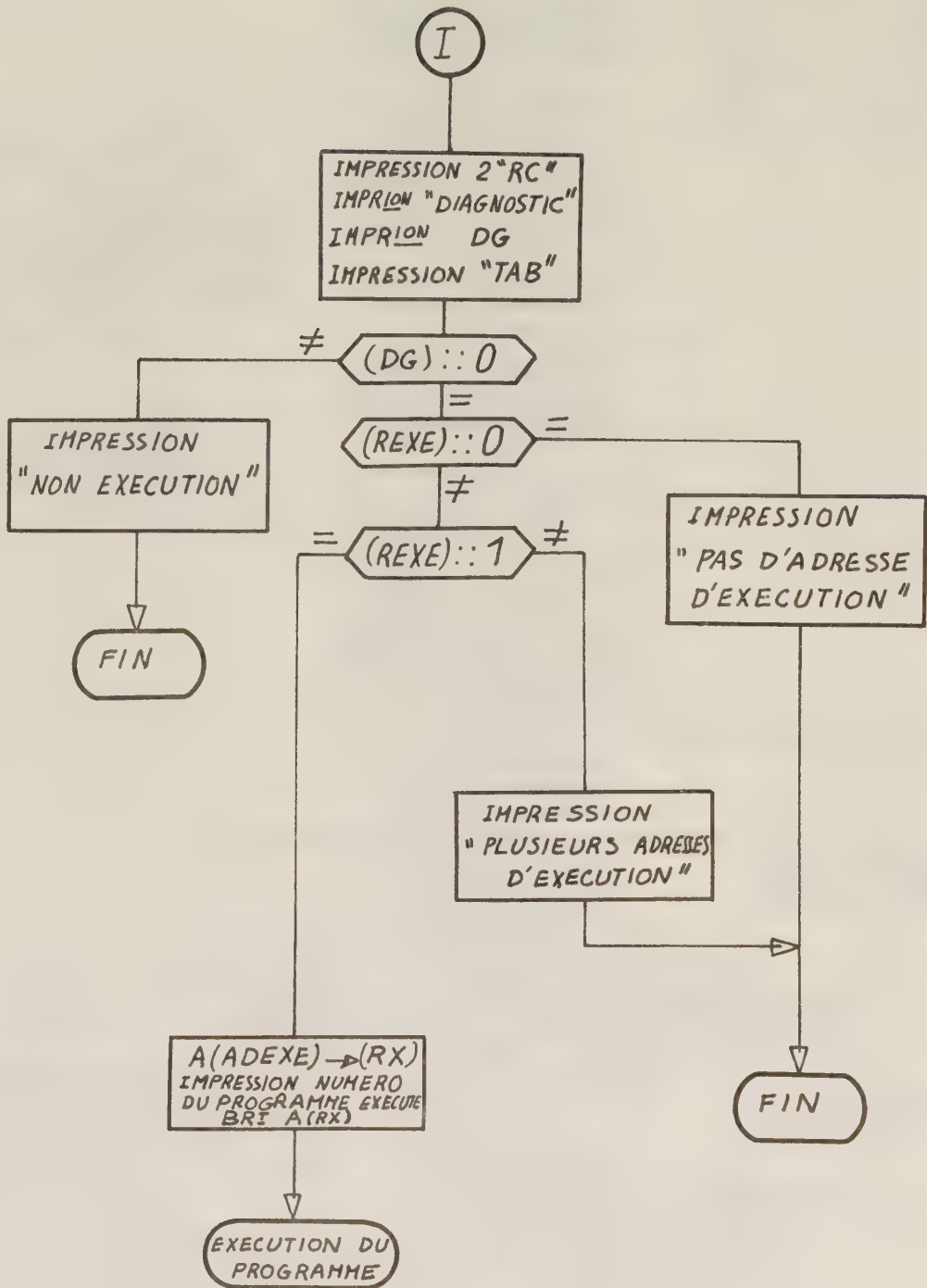
33- CHARGEUR P2 - ORG. Machine (4)



IMPRESSION TABLE des PROGRAMMES







EXECUTION du PROGRAMME .

37 - CHARGEUR P2 - ORG. Machine (8)

1	A8EMLAS X	ADRESSE DE CHARGEMENT
3	A2EM3AS X	A (DEFEX) → PTR2
5	A3EM5AS X	A (REFEX) → PTR3
	A7EVO	RAZ REXE, ADEXE, KP
	A15EM28Δ	
	T7EMO+	
10	A5EVO A0EL0S X	LECTURE
	V0TV32	TEST AVANCE-BANDE
	RV10AZ13	
	V0QV34	TEST B ₁₆ (DCB=34)
	FY14	
	RCV667Δ	LECTURE 4 CARACTERES
	T10R6	
	A13EM19AS	
19	X	ADRESSE MAXIMUM DE CHARGEMENT
	S13R8	TEST CHARGEABILITE
	FN13	
	RCV669Δ	LECTURE 8 CARACTERES
	T5R13	TEST NOMBRE MAXIMAL
	S13V10	DE PROGRAMMES
	FP13	
	K5QV1	2(KP)→(ACL2)
	A15EM28AS	
28	X	ADRESSE DE LA TABLE PROGRAMME
	A15R14	
	T10M1+	RANGEMENT NOM PROGRAMME
	T8R4	RANGEMENT ADRESSES LIMITEES
	A6R8	DU PROGRAMME
	K6QV15	
	A14R8	
	S14V32768	
	T14M2+	
	T6R8	
	A5V1	
40	T4R15 A0EL0S X	B - LECTURE
	I0TV40	TEST B4-B6
	RV64AY13	BRANCHEMENT C
	RCV669Δ	LECTURE 8 CARACTERES
	I0TV3	TEST B1-B2
	V13QV2	
	RV61AZ14	
	A11EV129024	CHARGEMENT RMSQ (MASQUE)
	A11W511	

	I10TW15872	TEST INSTRUCTION DE BRANCHEMENT
	V13QW9216	
	RV55AY14	
	K4TV11	
	RV57A	
55	C11QV7	
	K4TV4	
57	I10QR11	
	RV60AY0	
	RV61AZ14	
60	A10R13	TRANSLATION ADRESSE
61	T10EM0+	
	A15V1	
	RV40A	
64	V0TV8	C - TEST "8"
	RV131AZ13	
	V0TV35	TEST C ₁₆ (DCB=35)
	FY13	
	RCV669A	LECTURE 8 CARACTERES
	RV78AN10	TEST ADRESSE D'EXECUTION
	X	RESERVE
	I10QW1	TEST TRANSLATABILITE
	A10R4Z14	
	T15R12	TRANSLATION ET MISE EN PLACE
	A15EM28A	ADRESSE D'EXECUTION
	T10M0+	
	A7V1	
	T12R15	
78	A0ELOS	LECTURE
	X	
	V0QV37	TEST E ₁₆ (CARACTERE DE CONTROLE)
	FY14	
	RCV665A	D - LECTURE 2 CARACTERES
	T10R6	
	RV89AZ6	TEST EXISTENCE TABLE DES DEFINITIONS
	T2R13	EXTERNES
	A9EM3A	
	RCV107A	APPEL SP LECTURE TABLE
	T13R2	
89	A0ELOS	LECTURE
	X	
	V0QV38	TEST CARACTERE DE CONTROLE F ₁₆
	FY14	
	RCV665A	LECTURE 2 CARACTERES
	T10R6	
	RV100AZ6	TEST EXISTENCE REFERENCES EXTERNES
	T3R13	
	A9EM5A	
	RCV107A	SP LECTURE TABLE
	T13R3	

100	AOELOS	LECTURE
	X	
	VOTV32	TEST AVANCE-BANDE
	RV1(X)ΔZ13	
	VOTV36	TEST D16
	RV171ΔZ13	BT E
	RV14Δ	BT A
107	T13R15	SP LECTURE TABLE
	K5QV17	
	A14R15	MISE EN PLACE DU MOT DE CONTROLE
	A14W49152	
	A14V1	
	A14R6	
	T14M(X)+	
	A13V1	
	A9V384	
	T12R11	
117	T13R15	LECTURE 8 CARACTERES
	RCV669Δ	
	T10M(X)+	TRANSFERT DANS LA TABLE
	RCV667Δ	LECTURE 4 CARACTERES
	A10R4	TRANSLATION DE L'ADRESSE
	T10M1+	TRANSFERT DANS LA TABLE
	A13V2	TEST CAPACITE TABLE
	T13R14	
	S14R9	
	FP14	
	S6V2	TEST SORTIE SOUS-PROGRAMME
	RR11Z6	
	RR11N6	
	RV117Δ	
131	RCV669Δ	LECTURE 8 CARACTERES
	T10R6	
	I6TV98304	TEST CONSTANTES MULTIPLES ET
	S6R13	RESERVATION
	A15R6Z13	RESERVATION (EVOLUTION INDEX)
	RV4(X)ΔZ13	BT B
	RCV669Δ	LECTURE 8 CARACTERES
	T10R0	
	V13QV32768	TEST CONSTANTE SIMPLE OU DOUBLE LONGUEUR
	RV146ΔY14	
141	T0M(X)+	MISE EN PLACE CONSTANTE SIMPLE LONGUEUR
	A15V1	
	S6V1	
	RV4(X)ΔZ6	BT B
	RV141Δ	
146	RCV669Δ	LECTURE 8 CARACTERES
147	T0M(X)+	MISE EN PLACE CONSTANTE DOUBLE LONGUEUR
	T10M1+	
	A15V2	

	EV34	
	EV15	
	EV5	
	EV0	
	EV2	
	EV34	
	EV15	
	EV47	
	EV35	
	EV52	
	EV15	
	EV6	
	EV3	
	A1EV6	IMPRESSION DE LA DATE
214	EV30	
	S1V1	
	RV214ΔP1	
	A15EM218ΔS	
218	X	
	IEOM0+	
	EV11	
	IEOM1+	
	EV11	
	IEOM2+	
	EV30	IMPRESSION DU NUMERO DE PAGE
	EV1	
	EV25	
	EV25	
	EV25	
	A1EV8	MISE A JOUR COMPTEUR LIGNE
	A15EM28Δ	
	A6EV1	
	T6EM654Δ	MISE A JOUR COMPTEUR PAGE
	T6M402Δ	INITIALISATION COMPTEUR DIAGNOSTIC
	A0EV0	
	EV46	IMPRESSION SOUS-TITRE (MISE EN PAGE)
	EV47	
	EV23	NO.PR NOM DEBUT FIN
	EV10	
	EV24	
	EV48	
	EV50	
	EV30	
	EV46	
	EV47	
	EV45	
	EV30	
	EV30	
	EV36	
	EV37	

EV34	
EV53	
EV52	
EV30	
EV30	
EV38	
EV41	
EV46	
EV25	
EV25	
2 60	
A6V1	MISE A JOUR NUMERO DE PROGRAMME
IE5R6	IMPRESSION NUMERO DE PROGRAMME
EV30	
A10EM1+	APPEL NOM SYMBOLIQUE
RCV731Δ	IMPRESSION 5 CARACTERES DCB
EV30	
EV30	
A11EM2+	APPEL ET EXTRACTION ADRESSE DEBUT
I11QV32767	
T14R10	
RCV750Δ	IMPRESSION 4 CARACTERES HEXADECIMAUX
EV30	
EV30	
C11QV15	EXTRACTION ET IMPRESSION ADRESSE FIN
T11R10	
RCV750Δ	ECRITURE 5 CARACTERES H.
RCV637Δ	APPEL SP SAUT DE LIGNE
V6TR5	TEST FIN DE TABLE PROGRAMMES
A15V2Y13	
RV260ΔY13	
A15EM3Δ	TEST EXISTENCE TABLE DEFINITIONS EXTERNES
V15QR2	
A0EV1Z14	REXT=1
RV412ΔZ14	BT G
T1R14	TEST FIN DE PAGE
S14V43	
A1V3P14	
RCV635Δ	APPEL SP SAUT DE LIGNE
EV36	IMPRESSION TITRE
EV37	
EV38	DEFINITIONS EXTERNES
EV41	
EV46	
EV41	
EV52	
EV41	
EV47	
EV46	
EV51	
EV15	

	EV37	
	EV56	
	EV52	
	EV37	
	EV50	
	EV46	
	EV37	
	EV51	
308	T1R14	F - TEST SAUT DE PAGE
	S14V44	
	A1V2P14	
	RCV635Δ	APPEL SP SAUT DE LIGNE
	EV30	IMPRESSION SOUS-TITRE ET NUMERO
	EV48	DU PROGRAMME CHARGE
	EV50	PROGRAMME N
	EV47	
	EV39	
	EV50	
	EV33	
	EV45	
	EV45	
	EV37	
	A8EM0+	
	I8TV32767	
	C8QV17	
	I8QV16383	
	T13R8	
	T14R6	
	IE5R6	
	T15R6	
	A6V1	
331	T6R15	
	RCV637Δ	
	A4EV0	RDEF=0
	A11EM0+	IMPRESSION SYMBOLE
	T11R10	
	RCV731Δ	ECRITURE 5 CARACTERES DCB
	EV30	
	A10EM1+	IMPRESSION ADRESSE
	I10TW4	
	RCV750Δ	ECRITURE 4 CARACTERES H
	RV357ΔZ13	TEST MULTIDEFINITION
	EV30	IMPRESSION MESSAGE "MULTIDEF."
	EV15	
	EV15	
	EV45	
	EV53	
	EV44	
	EV52	
	EV41	

	EV36	
	EV37	
	EV38	
	EV23	
	EV10	
	EV24	
	RV369Δ	
357	V8TR2	TEST RN=PTR2 (DERNIER TABLEAU)
	RV369ΔZ13	
	T8R15	COMPARAISONS POUR DETECTION DES MULTIDEFINITIONS
	A15V1	
361	A9EM0+	
	RV366ΔN9	TEST MOT DE CONTROLE
	S9R11	
	RV376ΔZ9	TEST MULTIDEFINITION
	A15V1	
366	A15V1	
	V15QR2	TEST FIN DE COMPARAISONS POUR 1 SYMBOLE
	RV361ΔY14	
369	A6V2	
	V6QR8	TEST FIN DE SOUS-TABLEAU
	RV331ΔY14	
	V8QR2	TEST FIN DES COMPARAISONS
	T6R15Y14	
	RV308ΔY14	BT F
	RV412Δ	BT G
376	EV30	
	RV396ΔY4	TEST 1ERE MULTIDEFINITION
	EV23	ECRITURE MESSAGE "*MULTIDEF*"
	EV49	
	EV24	
	EV15	
	EV45	
	EV53	
	EV44	
	EV52	
	EV41	
	EV36	
	EV37	
	EV38	
	EV23	
	EV10	
	EV15	
	EV49	
	EV24	
	EV30	
396	A4V1	
	A9EM1+	EXTRACTION PARTIE ADRESSE
	I9TW4	ET IMPRESSION
	T9R10	

	RCV750Δ	ECRITURE 4 CARACTERES H.
	A14EM402ΔS	MISE A JOUR
402	X	DG (COMPTEUR DE DIAGNOSTICS)
	A14V1	
	T14M402Δ	
	RV408ΔY13	
	A9W4	MISE EN PLACE MARQUE DE MULTIDEFINITION
	T9M1+	
408	A15V2	TEST FIN DE COMPARAISONS POUR UN
	V15QR2	SYMBOLE
	RV369ΔZ14	
	RV361Δ	
412	A15EM5Δ	G - TRAITEMENT DES REFERENCES EXTERNES
	V15QR3	TEST EXISTENCE TABLE
	RV543ΔZ14	
	A4EM3Δ	
	A4V1	TEST SAUT DE PAGE
	T1R14	
	S14V43	
	A1V3P14	
	RCV635Δ	
	EV50	IMPRESSION TITRE
	EV37	REFERENCES EXTERNES
	EV38	
	EV37	
	EV50	
	EV37	
	EV46	
	EV35	
	EV37	
	EV51	
	EV15	
	EV37	
	EV56	
	EV52	
	EV37	
	EV50	
	EV46	
	EV37	
	EV51	
440	T1R14	TEST SAUT DE PAGE
	S14V44	
	A1V2P14	
	RCV635Δ	
	EV30	IMPRESSION SOUS-TITRE NUMERO DE
	EV48	PROGRAMME
	EV50	PROGRAMME N
	EV47	
	EV39	
	EV50	

	EV33	
	EV45	
	EV45	
	EV37	
	A8EMO+	
	I8TV32767	
	C8QV17	
	I8QV16383	
	T13R8	
	T14R6	
	IE5R6	
	T15R6	
	A6V1	
463	RCV637Δ	
	T6R15	EXTRACTION ET IMPRESSION SYMBOLE
	A11EMO+	
	T11R10	
	RCV731Δ	
	EV30	IMPRESSION PARTIE ADRESSE
	A10EM1+	
	RCV750Δ	
	EV30	
	RV485ΔY0	TEST DEFINITION (REXT=0)
	T4R15	
474	A9EMO+	
	RV480ΔP9	TEST MOT DE CONTROLE
	V15QR2	TEST FIN DU TABLEAU DEFEX
	A15V1Y14	
	RV474ΔY14	
	RV485Δ	
480	V9TR11	TEST DEFINITION (SYMBOLES EGAUX)
	RV505ΔZ13	
	A15V2	
	V15QR2	TEST FIN DU TABLEAU DEFEX
	RV474ΔY14	
485	EV30	IMPRESSION MESSAGE "*NON DEF.*"
	EV23	
	EV49	
	EV24	
	EV15	
	EV46	
	EV47	
	EV46	
	EV36	
	EV37	
	EV38	
	EV23	
	EV10	
	EV15	
	EV49	

	EV24	
	A14EM402Δ	MISE A JOUR COMPTEUR DIAGNOSTICS
	A14V1	
	T14M402Δ	
	RV530Δ	
505	EV30	EXTRACTION ET IMPRESSION ADRESSE DE
	A11EM1+	DEFINITION
	T11R10	
	RCV750Δ	
	T6R15	APPEL 1ERE ADRESSE A POURVOIR (ADRESSE
	A9EM1+	REFERENCE EXTERNE)
	T9R15	
512	A10EV129024	MISE EN PLACE DES ADRESSES DANS
	A10W511	LA CHAINE
	A9EM0+	
	I9TW15872	
	V13QW9216	
	C10TV7Y14	
	I10TR9	
	S9R13	
	RV524ΔY14	
	C13QV11	
	K11QV11	
	RV526Δ	
524	C13QV4	
	K11QV4	
526	A9R14	
	T9M0+	
	T13R15Y13	
	RV512ΔY13	
530	A6V2	TEST FIN D'UN SOUS-TABLEAU
	V6QR8	
	RV463ΔY14	
	V8QR3	TEST FIN DES COMPARAISONS
	RV579ΔZ14	BT I
	T6R15	
	RV440Δ	
	F	PLACE RESERVEE
	F	
	F	
	F	
	F	
	F	
543	RV579ΔY0	BT I
	RCV635Δ	
	EV36	IMPRESSION MESSAGE
	EV37	"DEFINITIONS EXTERNES NON UTILISEES"
	EV38	
	EV41	
	EV46	

EV41
EV52
EV41
EV47
EV46
EV51
EV15
EV37
EV56
EV52
EV37
EV50
EV46
EV37
EV51
EV15
EV46
EV47
EV46
EV15
EV53
EV52
EV41
EV44
EV41
EV51
EV37
EV37
EV51
RCV635Δ
EV36
EV41
EV33
EV39
EV46
EV47
EV51
EV52
EV41
EV35
EV30
A(EM4)2Δ
IE(RO)
RV8(X)ΔY0
RV8(X)ΔZ7
V7QV1
RV823ΔY14
A15EM28Δ
A(EM0)+
IQV32767

579

I - IMPRESSION MESSAGE

DIAGNOSTIC N

IMPRESSION NOMBRE DE DIAGNOSTICS
TEST EXECUTION POSSIBLE (PAS DE DIAGNOSTIC)
TEST DEFINITION ADRESSE D'EXECUTION
TEST MULTIDEFINITION ADRESSE D'EXECUTION

EXTRACTION ET IMPRESSION DE L'ADRESSE
D'EXECUTION DU PROGRAMME CHARGE

	COTV17	
	RCV635Δ	
	EV33	IMPRESSION MESSAGE
	EV36	
	EV50	ADRESSE D'EXECUTION <u>N</u>
	EV37	
	EV51	
	EV51	
	EV37	
	EV15	
	EV36	
	EV23	
	EV33	
	EV24	
	EV37	
	EV56	
	EV37	
	EV35	
	EV53	
	EV52	
	EV41	
	EV47	
	EV46	
	EV30	
	T14R10	
	RCV750Δ	ECRITURE 4 CARACTERES H.
	EV30	IMPRESSION "PR <u>M</u> "
	EV48	
	EV50	
	EV23	
	EV10	
	EV24	
	IELR0	NO. DU PROGRAMME EXECUTE
	RRL5	EXECUTION
	F	RESERVE
635	EV25	SP SAUT DE LIGNE
	A1V1	
637	EV25	
	A1V1	
	T12M640ΔS	
640	X	
	S1V46	
	RV645ΔP1	
	A1V46	
	RM640Δ	
645	A1V3	
646	EV25	
	S1V1	
	RV646ΔP1	
	A1EV13	

650	EV30 S1V1 RV650AP1 A1EM654AS	
654	X A1V1 IEOR1 T1M654A A1EV1 EV25 EV25 EV25 RM640A F F	COMPTEUR DE PAGES RESERVE
665	A14EV0 RV710A	SP LECTURE HEXADECIMALE
667	A14EV0 RV696A	4 CARACTERES
669	A10ELOS X S10V33 A10V23N10 A10V10 K10QV4 A10ELOS X S10V33 A10V23N10 A10V10 A10R14 K10QV4 A10ELOS X S10V33 A10V23N10 A10V10 A10R14 K10QV4 A10ELOS X S10V33 A10V23N10 A10V10 A10R14 K10QV4 A10ELOS X S10V33 A10V23N10 A10V10 A10R14 K10QV4 A10ELOS X S10V33 A10V23N10	8 CARACTERES
696		

	A10V10	
	A10R14	
	K10QV4	
	A10ELOS	
	X	
	S10V33	
	A10V23N10	
	A10V10	
	A10R14	
710	K10QV4	
	A10ELOS	
	X	
	S10V33	
	A10V23N10	
	A10V10	
	A10R14	
	K10QV4	
	A10ELOS	
	X	
	S10V33	
	A10V23N10	
	A10V10	
	A10R14	
	A10W32768D10	MISE EN PLACE DIGIT DE "DEBORDEMENT"
	RR12	
	K10QV19	SP ECRITURE SOUS FORME ALPHANUMERIQUE
	RV741Δ	
	K10QV13	
	RV738Δ	
	K10QV7	
	RV735Δ	
731	K10QV1	5 CARACTERES
	T14R10	
	C10QV25	
	ER10	
	T14R10	
	C10QV25	
	ER10	
	T14R10	
	C10QV25	
	ER10	
	T14R10	
	C10QV25	
	ER10	
	T14R10	
	C10QV25	
	ER10	
	RR12	
	K10QV23	SP ECRITURE SOUS FORME HEXADECIMALE
	RV787Δ	

750	K10QV15 RV775Δ C10QV28 S10V10 A10V23P10 A10V10 ER10 T14R10 C10QV27 S10V10 A10V23P10 A10V10 ER10 T14R10 C10QV27 S10V10 A10V23P10 A10V10 ER10 T14R10 C10QV27 S10V10 A10V23P10 A10V10 ER10	4	CARACTERES
775	T14R10 C10QV27 S10V10 A10V23P10 A10V10 ER10 T14R10 C10QV27 S10V10 A10V23P10 A10V10 ER10 T14R10 C10QV27 S10V10 A10V23P10 A10V10 ER10 T14R10 C10QV27 S10V10 A10V23P10 A10V10 ER10 T14R10 C10QV27 S10V10 A10V23P10 A10V10 ER10 RR12		

800	EV30 EV46 EV47 EV46 EV15 RV847Δ	ECRITURE "NON"
806	EV25 EV48 EV33 EV51 EV15 EV36 EV23 EV33 EV24 EV33 EV36 EV50 EV37 EV51 EV51 EV37 RV842Δ	ECRITURE "PAS D'ADRESSE"
823	EV25 EV48 EV44 EV53 EV51 EV41 EV37 EV53 EV50 EV51 EV15 EV33 EV36 EV50 EV37 EV51 EV51 EV37 EV51	ECRITURE "PLUSIEURS ADRESSES"
842	EV15 EV36 EV23 EV33 EV24	ECRITURE "D'EXECUTION"
847	EV37 EV56 EV37	

	EV35	
	EV53	
	EV52	
	EV41	
	EV47	
	EV46	
	EV25	
857	F	ARRET ET FIN CHARGEMENT
	RV857Δ	
	F	RESERVE
	FE	FIN BANDE PROGRAMME

CHAPITRE 5

MISE EN PLACE ET ESSAIS DU SYSTEME

I - OCCUPATION DE LA MEMOIRE CENTRALE

Lors des généralités nous avons constaté la faiblesse des entrées-sorties pour le modèle d'ordinateur pris comme exemple de notre étude. L'absence de mémoire auxiliaire et la lenteur des éléments d'entrée oblige à laisser en permanence en mémoire centrale les divers programmes constituant le système d'exploitation (à titre d'exemple, la lecture du ruban codé correspondant au compilateur PAF nécessite environ 15 minutes...).

1 - Encombrement à prévoir

- a) - Chargeur : 800 instructions (moniteur et chargeur proprement dit).
- b) - Assembleur : 3305 instructions comprenant les cinq segments de programme, les tables de données utilisées et les directives.
- c) - Le système d'assemblage - chargement occupe donc pour la partie programme 4105 instructions soit sensiblement 4 K mots.

A cela il faut ajouter les tables à constituer en cours de travail :

- Table programme source (DCB)

D'après les essais réalisés (cf. paragraphe III), la taille prise par le programme source en mémoire est en moyenne de cinq mots par ligne de programmation (4 mots environ pour les programmes sans commentaires,

de 7 à 8 mots pour des sections comportant des commentaires à chaque ligne).

- Table programme objet binaire

Instructions et constantes générées, réservation de mémoires.

- Table des codes instruction

Il a déjà été dit que chaque instruction ou chaque directive définit un code instruction à 4 digits binaires. Chaque mot contenant 8 de ces champs, il faut prévoir sensiblement $N/8$ positions de mémoire pour cette table, N étant le nombre de lignes de programmation (en toute rigueur, c'est le nombre d'instructions qui intervient).

- Tables des symboles : symboles, définitions et références externes.

Voyons les configurations réalisables en fonction des deux tailles possibles de mémoire pour la série CAB 500.

2 - Modèle A (capacité 16 K)

La géographie de la mémoire centrale est celle de la figure 21a

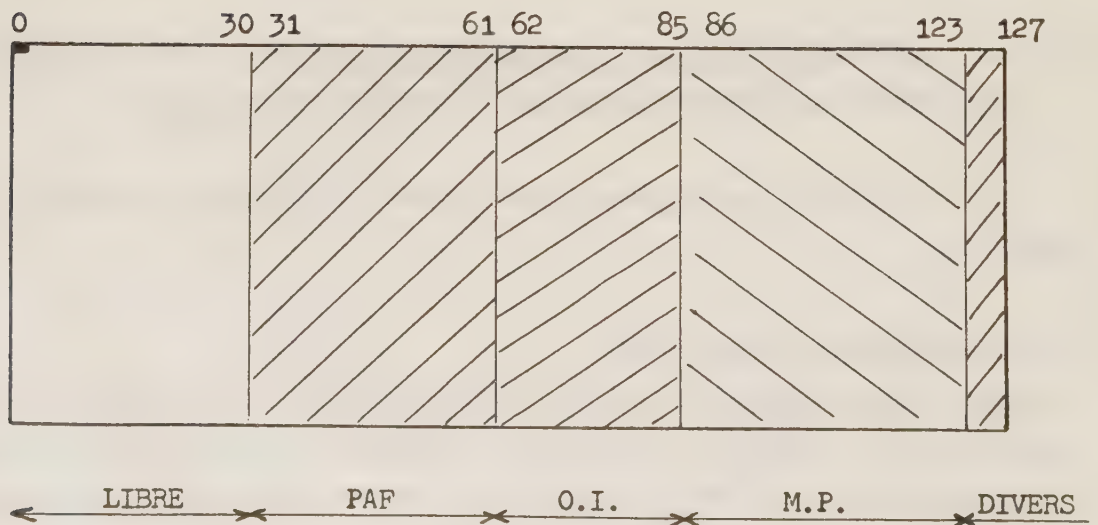


Figure 21a

Sans modifier l'implantation actuelle, la place libre est d'environ 4 K ; afin de ne pas réduire considérablement la taille des tables, nous utilisons la technique suivante :

- Le chargeur réside constamment en mémoire centrale.
- Pour un travail mixte l'assembleur est placé en mémoire centrale à partir du chargeur (temps de l'opération : environ 10 minutes).
- La zone contenant le compilateur PAF est mise à disposition, tandis que les ordres initiaux sont conservés car ils contiennent certains programmes de mise au point intéressants ; la zone des microprogrammes n'est évidemment pas utilisable. La dimension totale est de 7915 positions (62 pistes).

a) - Cas du chargement

Table des définitions externes	384 positions
Table des références externes	384 positions
Table des programmes chargés	12 positions
Chargeur	800 positions
	1580 positions

La place disponible pour loger les programmes à charger représente un peu plus de 6 K mots (6335 exactement).

b) - Cas de l'assemblage

Table des symboles	256 positions
Table des définitions externes	256 positions
Table des références externes	256 positions
Assembleur	3305 positions
	4073 positions

Le programme source et le programme binaire avec ses codes instruction doivent occuper au maximum 3842 mots, la table du programme généré s'obtient donc par la relation :

$$T = 3842 - 5,125 N \text{ avec } T > N \quad (1)$$

(cf. figure 22).

Un segment utilisant peu de constantes et variables peut avoir une taille maximale de 600 instructions environ, un petit programme au contraire pourra faire appel à des réservations importantes.

Du point de vue utilisation, dans la mesure des 6 K mots disponibles lors du chargement, un programme important utilisant par ailleurs des tableaux de grande taille peut être assemblé en plusieurs segments, dont l'un comportant peu d'instructions exécutables définit toutes les variables et constantes.

3 - Modèle B (capacité 32 K)

La disposition en mémoire centrale est similaire (figure 21b)

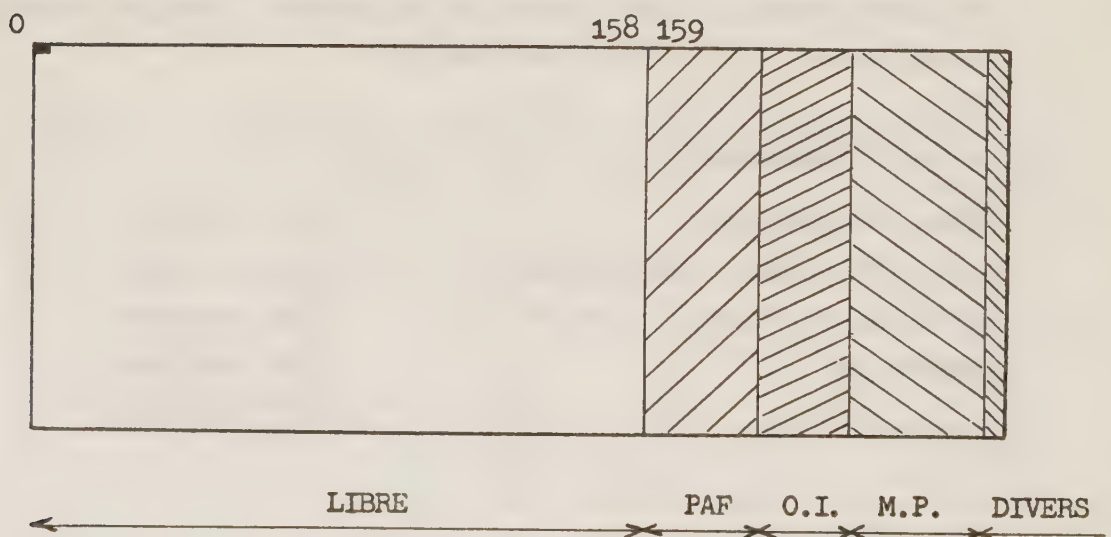


Figure 21b

La place libre en mémoire passe à 20 K mots (159 pistes), le compilateur PAF, assembleur et chargeur peuvent résider constamment en mémoire centrale.

a) - Chargement : le système utilisant 4885 positions les programmes à charger peuvent occuper plus de 15 K .

b) - Assemblage : la taille du segment à assembler est donnée par l'expression :

$$\underline{T = 15480 - 5,125 N} \quad \text{avec } T > N \quad (2)$$

Les relations (1) et (2) sont représentées graphiquement sur la figure 22.

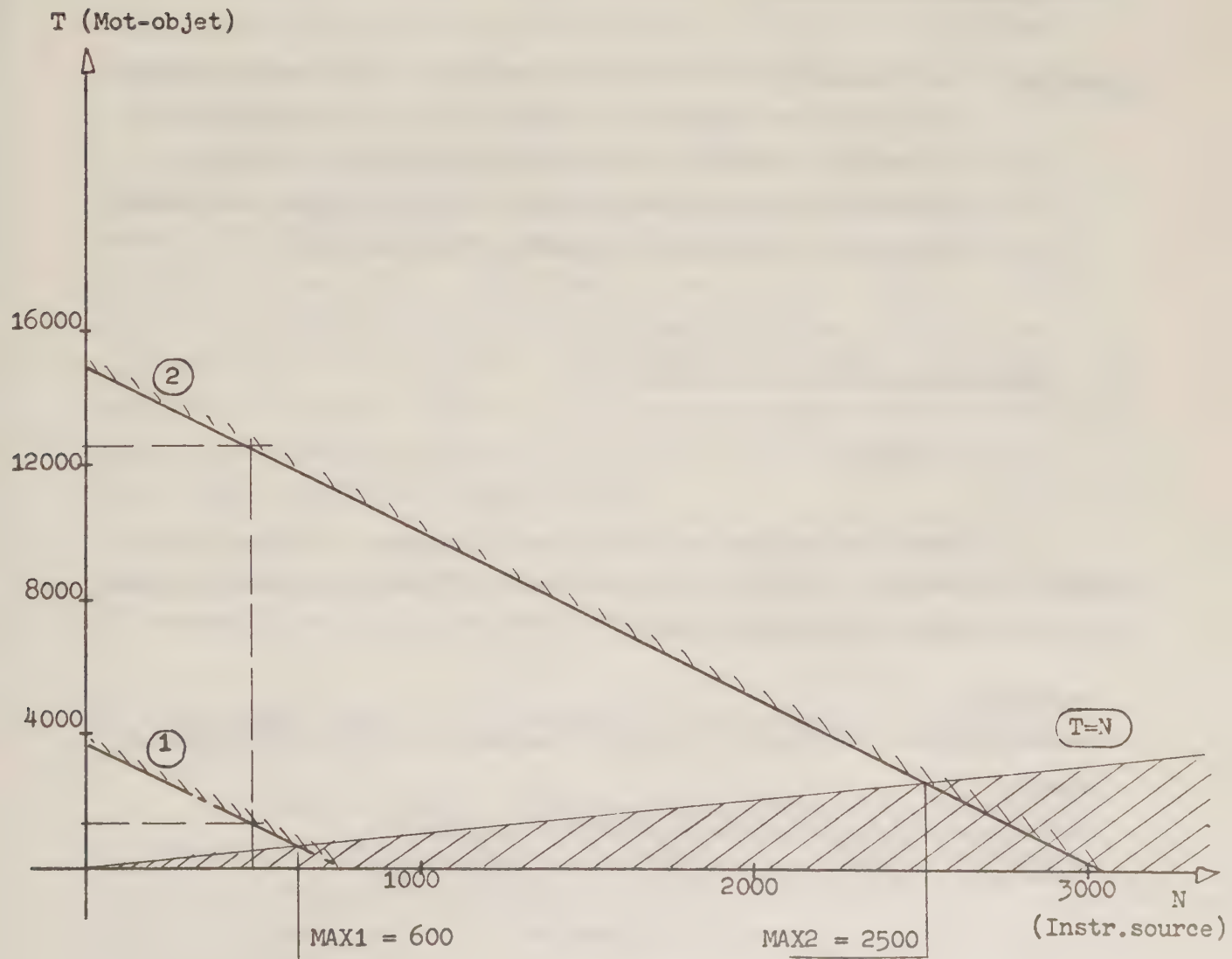


Figure 22

En conclusion

Les possibilités apportées en multipliant par deux la capacité de la mémoire sont bien plus que doublées. Le nombre maximum d'instructions source à assembler, passe de 600 à 2500, tandis que les segments à charger peuvent occuper 15 K au lieu de 6 K.

A titre d'exemple :

Dans le premier cas, un programme de 500 instructions source peut engendrer jusqu'à 1200 mots objet alors que dans le 2ème cas, le maximum est de 12500. Si l'on veut envisager l'extension du système au macro-assemblage, cela n'est évidemment possible que dans le cas d'une capacité mémoire centrale de 32 K ; la taille du système devant nécessairement augmenter, il ne resterait plus de place disponible pour loger le programme à assembler.

Remarque : Les dimensions des tables de symboles ont été choisies dans des limites qui semblent en rapport avec les possibilités de la machine, toutefois, nous verrons dans les conclusions, l'amélioration permettant d'avoir une utilisation plus souple du système. (Gestion plus dynamique des tables).

II - UTILISATION PRATIQUE DU SYSTEME

1 - Rappel

L'appel au système est obtenu par la frappe au clavier de la suite des symboles : $\Delta \Delta$. Suivant les deux lettres placées derrière, le fonctionnement est orienté sur :

- chargement
- assemblage avec édition de la liste d'assemblage et de la bande objet binaire
- assemblage et impression de la liste correspondante.
- assemblage et perforation de la bande binaire objet.

2 - Assemblage

La bande source étant positionnée en lecture, l'utilisateur n'a pas d'autre manipulation à effectuer en cours de travail.

Le cas échéant, le système demande si un organe de perforation est bien connecté, après réponse affirmative du programmeur (par OK) la bande binaire objet est générée.

3 - Chargement

En cours de chargement le problème se pose différemment car, a priori, le système ne connaît pas le nombre de segments de programme à placer en mémoire. La fin de bande est repérée par une succession de codes "avance-bande" qui ne spécifient pas s'il s'agit de la dernière section à charger.

Après rencontre de ces codes, la machine est positionnée en "attente lecture" et c'est l'utilisateur qui oriente la suite du traitement. S'il fournit une autre bande en lecture, le premier code significatif rencontré ($11_{10} = B_{16}$) renvoie à une nouvelle séquence de chargement, s'il s'agit au contraire du dernier programme, le programmeur l'indiquera par le code "fin de chargement" ($13_{10} = D_{16}$), soit en frappant au clavier la lettre D, soit en ayant perforé préalablement un D à la fin de la dernière bande lue (ou sur une bande comportant cette lettre comme premier code significatif).

Dans les deux cas, l'opération suivante est l'édition de l'état de chargement que nous avons décrit dans le chapitre 4.

III - PROGRAMMES TESTS : (cf. programmes tests et essais en annexe)

Afin de vérifier le fonctionnement correct de l'assembleur et du chargeur, un certain nombre de programmes ont été écrits en langage d'assemblage et passés en machine (ceux-ci ont été utilisés pour la mise au point du système global). Ces programmes ne représentent rien du point de vue exécution.

1 - Programmes de test de l'assembleur

Chacun des programmes a un rôle particulier

- a) - Le programme TEST 1 vérifie l'assemblage correct des types d'instructions et des opérandes. Les sections traitant les étiquettes, les codes opération et les opérandes étant indépendantes il n'est pas utile de tester toutes les combinaisons possibles, les possibilités dans chacune des séquences de traitement suffisent.
- Les étiquettes et symboles appartiennent aux trois types possibles :
 - symboles normalement définis (ALPHA, BETA, JUMPI, RAD, TEST 1) ;
TEST 1 a un rôle particulier puisqu'il apparaît en 1ère instruction du programme (compteur d'adresse nul), il indique le nom du programme.
 - définitions externes : GAMMA, JOK
 - références externes : ARC, TABEX

Il faut remarquer à ce sujet, la liaison en chaîne pour cette dernière (TABEX 024 → 01C → 008). L'utilisation de la méthode de la liste permet d'attribuer aux instructions n° 23 et 27 (emplacement 12 et 16) l'adresse 017 pour JUMP1 qui est défini par la suite (instruction n° 28).

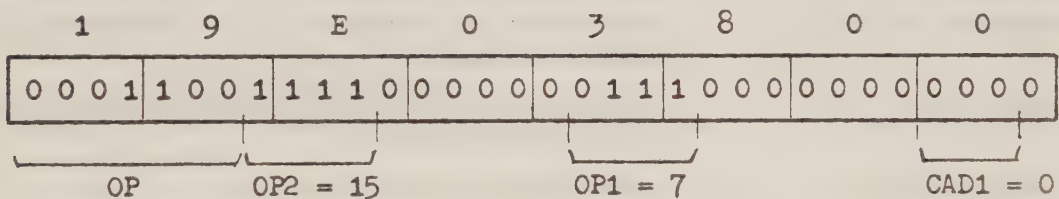
- Les codes opération ont été choisis dans chaque classe d'instructions définie dans l'étude de l'assembleur (annexes n° 2 et 3) pour vérifier l'exactitude de la translittération.
- Les opérandes sont aussi de tous les types : registre, immédiat décimal ou hexadécimal (avec ou sans facteur d'échelle), symbolique (expressions arithmétiques, auto-indexation).

Le code objet est conforme à la description du langage machine fournie dans le chapitre I - III. Le programme ne comportant pas d'erreur (pas de diagnostic) la lecture de la bande binaire générée permet de vérifier la conformité de cette liste (code début, nombre d'instructions, nom symbolique, instructions générées, code fin de programme, tables des définitions et références externes).

- b) - Le programme TEST 2 montre que toutes les opérations exécutables sont traduites normalement en langage machine. Les différentes possibilités d'opérandes ayant été vérifiées dans le segment TEST 1, chaque instruction fait intervenir un (ou des) opérande (s) en registre.

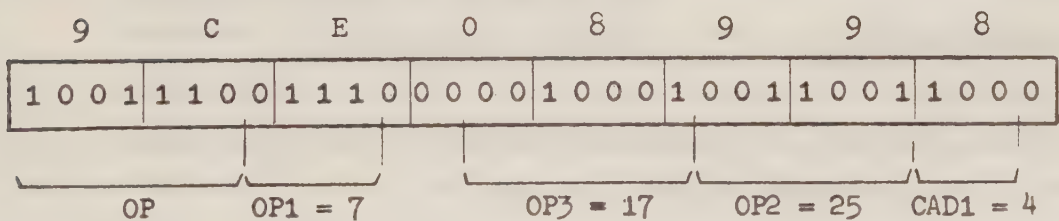
A titre d'exemple, voyons les codes objet correspondant aux instructions : TR 7,15 et IMLR 7,25,17 (figure 23).

Transfert : TR 7,15



Impression d'un nombre décimal exprimé en double longueur :

IMLR 7,25,17



- Figure 23 -

- c) - Les programmes TEST 3, TEST 5 et TEST 6 sont tout à fait similaires, les différences venant surtout des symboles utilisés ; TEST5 et TEST6 ont été créés pour être chargés avec d'autres segments de manière à réaliser les essais du chargeur.

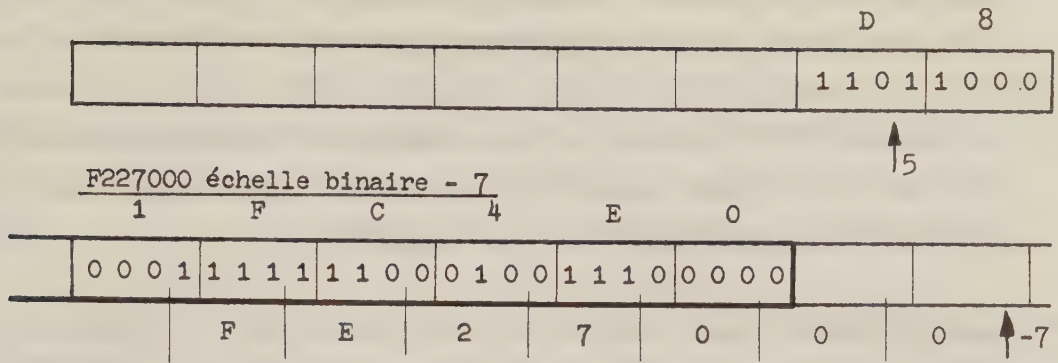
TEST3 envisage les diverses directives de réservation et de génération de données :

instructions 6 et 7	: réservation de mémoire
instruction 8	: définition de constante fixe décimale
instructions 9 et 10	: définition de constante fixe avec échelle binaire
instruction 11	: définition de constante fixe hexadécimale
instruction 12	: définition de constante fixe hexadécimale avec échelle binaire
instruction 13	: définition de constante fixe décimale avec facteur multiplicatif
instruction 14	: constante en simple longueur
instruction 15	: constante en simple longueur facteur multiplicatif et échelle décimale
instruction 16	: constante en double longueur
instructions 17 et 18	: constante en double longueur avec facteur multiplicatif et échelle décimale
instructions 19 et 20	: chaînes de caractères.

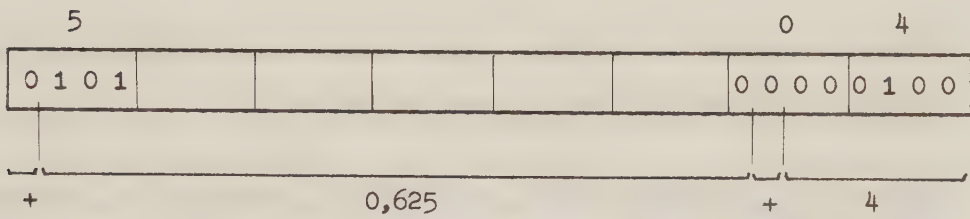
A titre d'exemple la représentation binaire des instruction 10, 12, 14, 16 et 20 est donnée par la figure 24.

6,75 échelle binaire 5

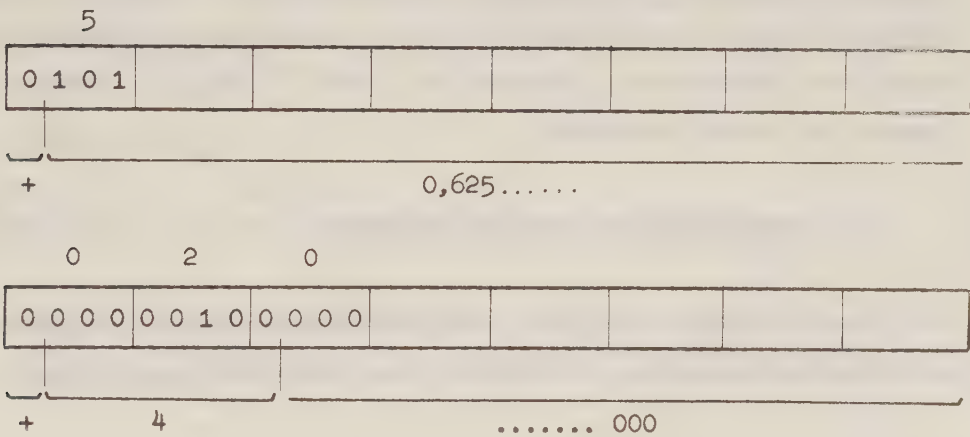
$$(6,75)_{10} = (110,11)_2$$



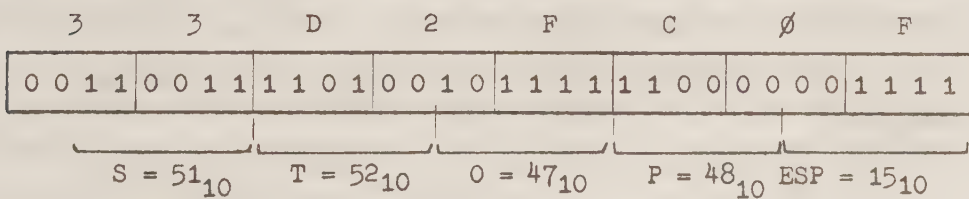
10 en simple longueur - $(0,625 \times 2^4)$; taille $(100)_2$, figure $(101)_2$



10 en double longueur



STOP chaîne de caractères - complétée à 5 par code 15₁₀ (ESPACE)



Le programme ne comporte que des symboles normalement définis : BETA, F3, GAMMA et TEST3 (nom symboliques du programme).

- d) - Le segment PTEST4 montre la façon dont sont traitées et mises en évidence les erreurs rencontrées en cours d'assemblage. Chaque type d'erreur étant représenté dans ce programme.

Dans le cas d'erreur dans la zone étiquette seule, l'instruction est assemblée ; dans le cas contraire à la place de celle-ci apparaît le diagnostic, la valeur actuelle du compteur d'adresse n'étant pas imprimée.

(Pour la clarté de la liste, nous avons placé entre deux instructions erronées une instruction de non opération).

Notons que le symbole BETA, multidéfini, conserve sa première adresse de définition (0009) et que le symbole TABEX utilisé par une instruction arithmétique est considéré comme une référence externe au segment assemblé, la rencontre ultérieure de l'étiquette correspondante est donc une multidéfinition et entraîne un diagnostic. Le total des diagnostics, imprimé en fin de "listing", indique le nombre de lignes erronées.

2 - Programmes de test du chargeur

Les programmes de test vus précédemment sont groupés différemment et chargés de manière à décrire les divers cas possibles.

a) - TEST1

Le chargement du programme TEST1 seul conduit au refus d'exécution car celui-ci contient deux références externes (ARC 0A25 - TABEX CA24) qui ne sont donc pas définies par ailleurs (le chaînage ne peut être réalisé).

L'adresse de base, fixée par le système, étant 0A00, les adresses réelles des divers symboles sont 0A23, 0A12, 0A25 et 0A24. Les diagnostics de non définition apparaissent effectivement en face de chaque référence correspondante.

b) - TEST2

Le programme TEST2 utilisé seul, n'entraîne pas de diagnostic d'erreur au chargement. L'exécution n'est pas validée parce que ce segment ne définit pas d'adresse d'exécution (opérande de la pseudo-instruction fin).

c) - TEST1, TEST2, TEST3

Le chargement des trois programmes assemblés séparément ne modifie pas le résultat final puisque les sections 1, 2 et 3 ne comportent aucun symbole commun ; le chargement s'effectue en plaçant chaque section à la suite l'une de l'autre. La séparation de la zone de travail en deux parties (données et programmes) ne présente pas d'intérêt puisqu'il n'y a pas de protection mémoire programmable sur la machine considérée.

d) - TEST1, TEST6, TEST5

Le groupement de ces sections mène à des multidéfinitions.

Le programme 1 définit les références externes contenues dans les programmes 2 et 3 (GAMMA 0A23, JOK 0A12).

Par ailleurs le segment n°2 définit : ARC (0A29), COMPT (0A5B) et TABEX (0A6F), tandis que la section n°3 redéfinit les symboles ARC (0A92) et TABEX (0AD8).

Les diagnostics de multidéfinition sont placés vis à vis de ces définitions dans les programmes 2 et 3. Pour la première section les utilisant on trouve aussi l'adresse de la seconde définition (0A92 pour ARC, 0AD8 pour TABEX).

Les références externes ARC et TABEX citées dans le programme 1, ont pour adresse celle de la première définition. Ceci permet de reconstituer facilement les liaisons.

En effet le programmeur, à la lecture de l'état de chargement, constate que les segments 2 et 3 définissent des entrées utilisant le même nom symbolique. Sur la même ligne, il voit les adresses de ces autres définitions et peut à partir de la table des programmes, savoir dans quelles sections elles se trouvent.

e) - TEST1 et TEST6

Les segments se complètent normalement, chacun définit les références externes utilisées par l'autre et réciproquement.

C'est dans la table des références externes que l'on retrouve l'adresse d'utilisation et l'adresse de définition.

Le programme chargé n'est toutefois pas exécutable car aucune des sections ne comporte d'adresse définissant le début du segment.

f) - TEST5, TEST1

Le chargement des sections TEST5 et TEST1 conduit à un programme théoriquement exécutable, aucun diagnostic n'étant signalé. L'exécution a évidemment été stoppée puisque le programme chargé ne correspond à rien.

L'adresse d'exécution est fournie par le programme TEST5 (adresse 0A67).

IV - COMPARAISON AVEC L'ANCIEN SYSTEME

Nous rappelons que le langage autocode utilisable avec les "ordres initiaux" ne permet pas l'utilisation de symboles en partie adresse des instructions. Toutes les adresses sont donc numériques, en cas de modifications ou de segmentation de programme, l'utilisation devient très laborieuse et les risques d'erreur sont multiples. A la lecture d'un programme autocode, celui-ci est traduit en binaire et chargé instruction par instruction. Il existe un code binaire spécial dit "code S" qui est implanté plus rapidement en mémoire ; c'est le programme en absolu qui est fourni par la machine sous forme de 6 caractères par mot, le premier ne comportant que 3 digits (33 bits par mot).

1 - Essais réalisés

Les quatre types de programmes tests décrits précédemment, ont été assemblés et chargés séparément tandis que les temps des diverses opérations ont été mesurés. La machine ne possédant pas de système de gestion évolué, la durée du travail de la mémoire centrale est inconnue et les temps obtenus sont ceux du traitement complet (avec entrée-sortie) ; ils ne fournissent donc pas une idée très exacte des performances du software.

A un programme ainsi placé en mémoire correspond, sauf impossibilité, une forme autocode et une forme codée S ; elles ont été sorties sur bande perforée à partir des combinaisons de commande de l'ancien système, les temps de lecture de ces bandes fournissent des éléments de comparaison par rapport au chargeur conçu. Toutefois il faut remarquer que les programmes tests utilisent des symboles (références et définitions externes) qui constituent autant de renseignements à fournir au chargeur, allongeant ainsi les temps de traitement. Dans le cas de l'autocode tous les travaux dus à l'affectation des adresses doivent être faits par le programmeur d'où des pertes de temps considérables.

2 - Résultats et interprétations "des essais"

Le tableau en annexe n° 10 fournit les résultats des diverses mesures réalisées.

a) - Voyons d'abord les cas particuliers des programmes TEST3 et TEST4 :

L'assemblage du programme TEST4 qui montre les différents types d'erreur ne fournit pas de ruban binaire qui n'aurait aucun sens, il n'est donc pas possible de relever de temps de chargement ; par ailleurs, il ne correspond aucun programme similaire dans l'ancien système, car la rencontre d'une erreur de syntaxe se traduit suivant les cas, par le blocage de la machine ou la génération d'une instruction binaire erronée dont les conséquences sont plus ou moins graves lors de l'exécution du programme chargé.

De même il ne correspond pas de réservation de mémoire et de pseudos-instructions telles que celles rencontrées dans le segment TEST3 ; il est possible de charger des constantes à partir de combinaisons de commandes mais cela est assez laborieux et long. La comparaison n'est pas tellement significative car les méthodes sont vraiment différentes.

b) - Les temps de sortie des listes (assemblage et binaire) sont très élevés, du fait de la lenteur des périphériques.

Le temps d'écriture de la liste d'assemblage est évidemment proportionnel au nombre de mots du programme source (1,38 secondes par mot) et au nombre de lignes de programme (8,6 s/ligne). L'édition du programme TEST2 qui compte 80 lignes comportant des commentaires nécessite 840 secondes.

Le temps de perforation de la bande binaire est lié plus directement au nombre d'instructions générées (1,65 s/instruction), il est très diminué dans le cas des directives (0,14 s) car le procédé de traitement des constantes multiples est assez performant (cf. chapitre 3 paragraphe VIII - 1).

c) - Le temps d'assemblage est plus significatif des performances du système puisqu'il ne fait intervenir l'organe d'entrée que pendant la lecture de la bande (50 cre/s), il est fonction du nombre d'instructions assemblées (1,5 s/i) mais aussi de la forme de celles-ci (présence de commentaires).

d) - Le temps de chargement dépend fortement du type de programme assemblé :

0,26 s/i pour TEST1 qui comporte des tables de références et définitions externes.

0,18 s/i pour TEST2 qui ne comporte que des instructions simples.

0,08 s/i pour TEST3.

Comparons au chargement avec l'ancien système :

en autocode 0,33 s/i pour TEST1 - 0,3 pour TEST2

en code S 0,21 s/i pour TEST1 - 0,22 pour TEST2.

Il est normal que pour ce système, le temps de chargement d'une instruction soit sensiblement constant puisqu'il s'agit de programmes similaires. Aucun symbole n'intervenant, la différence vient du type des opérandes (TEST2 ne comporte que des opérandes en registre).

Nous remarquons toutefois que le chargement est sensiblement aussi performant que le "code S" et meilleur que l'autocode. Le code S ne comporte pourtant que 6 caractères par instruction au lieu de 9 (code instruction plus 8 codes hexadécimaux).

La moyenne des temps correspondant à l'assemblage et au chargement des trois programmes TEST1, TEST2, TEST3 donne les performances d'un programme comportant à la fois directives et commentaires :

(T ASS) moyen	= 1 s/i
(T LIS.B.) moyen	= 1,3 s/i
(T CHt) moyen	= 0,18 s/i

Nous constatons donc que le temps de traitement avant exécution est trois fois plus long pour un programme rédigé en langage symbolique. Toutefois celui-ci présente deux importants avantages par rapport au langage autocode :

- Gain de temps à l'écriture et à la mise au point des programmes (utilisation de symboles, directives, segmentation, listes d'assemblage et de chargement).
- Réduction considérable des risques d'erreur en particulier dans l'affectation des adresses qui est faite par le système (symboles).

Remarque : les ordres initiaux sont capables d'effectuer la translation d'adresses numériques pour un programme autocode rédigé dans cette optique mais les programmes dits "relatifs" sortis sur bande par le software fourni par le constructeur ne sont pas relogeables car le type des adresses n'est pas reconnu par le système de liste binaire ; seules, les instructions de branchement sont systématiquement considérées comme relatives à l'adresse de chargement (ce qui n'est pas toujours le cas).

CONCLUSIONS

Le système fourni par le constructeur de l'ordinateur CAB500 permet d'utiliser deux langages de programmation très différents : un langage autocode proche du langage machine et le PAF pour les calculs scientifiques.

Certains programmes et sous-programmes répétitifs ont intérêt à être réalisés à partir d'un langage plus performant que le PAF. Par ailleurs, celui-ci ne permet pas de réaliser des travaux d'édition (pas de possibilité d'impression de titres, tableaux....).

Le langage autocode répond à ces besoins mais la programmation est fastidieuse et difficile (absence de symboles, de listes et de diagnostics). Nous avons pensé qu'il était intéressant de créer un langage d'assemblage suffisamment proche de cet autocode pour ne pas perdre en performances ce que l'on gagne en puissance de rédaction des programmes. Un système d'assemblage et de chargement correspondant à ce langage symbolique assure la translittération des instructions et l'exécution des programmes tout en éditant des listes qui facilitent le travail du programmeur (surtout en période de mise au point).

L'ensemble a été conçu et programmé de façon à rester le plus souple possible en permettant de faire certaines améliorations sans modification importante et complexe des sections déjà en place. En particulier, il peut être avantageux de ne pas fixer à priori la taille de chaque table élémentaire utilisée au cours de l'assemblage ou du chargement, mais de prendre une zone globale pour toutes les tables, le système gérant le partage de cette zone en fonction des besoins.

Les diverses adresses des tables sont des données fournies aux pointeurs en cours de travail à partir de mémoires particulières qui peuvent être modifiées sans problème à n'importe quel moment de l'assemblage ou du chargement. Il est donc aisé de placer à la suite les unes des autres, les tables qui ne sont pas créées simultanément; la première adresse libre à partir de la dernière table constituée initialise le pointeur du tableau qui suit. Pour l'assemblage, on loge à la suite table du programme source et table des symboles, le reste de la place disponible en mémoire est réservée aux instructions binaires générées.

Par ailleurs, nous avons prévu dans la phase d'assemblage, un débranchement pour le traitement de macro-instructions ; cette partie n'est pas réalisée, car nous pensons que cela alourdirait considérablement le système qui est déjà lent par la nature même de la machine. Malgré les précautions prises et les solutions adoptées pour limiter les temps d'assemblage et de chargement, l'ensemble de ceux-ci est sensiblement cinq fois plus important que pour un programme écrit en autocode. Bien que difficile à évaluer, le gain de temps à la mise au point compense largement cette perte. Du point de vue de l'exécution des programmes, les temps sont pratiquement identiques puisque le langage symbolique est très proche de l'autocode ; dans le cas où les quelques restrictions permettant de clarifier les instructions, allongeraient le temps de calcul, il est possible, pour gagner du temps dans des programmes répétitifs, de rédiger ceux-ci en autocode à partir de la liste fournie par l'assembleur.

Bien que la CAB500 soit un matériel démodé, le nombre d'utilisateurs est encore assez élevé et cette étude présente un exemple de système intéressant pour un petit ordinateur, certaines méthodes de traitement pouvant être avantageusement utilisées. Les performances seraient considérablement accrues par un ordinateur ayant des caractéristiques plus favorables en particulier du point de vue temps d'accès à la mémoire centrale.

BIBLIOGRAPHIE

Machines de Traitement de l'Information (Circuits et Programmes)
par P. DEBRAINE.

Tome 1 - Etude Logique et Construction des circuits

Tome 2 - Programmation : Principes et Langages d'assemblage

Tome 3 - Langages algorithmiques et systèmes de programmes

chez MASSON (Monographies d'Electronique).

Computer Programming

Computer Software

by Ivan Flores (Prentice-Hall - series in applied Mathematics).

Programming and coding for automatical digital computers by George W Evans II
and Clay L. Perry (Mc Graw-Hill).

Programming and coding digital computers by Philip M. Sherman
(John Wiley and Sons - Inc).

Programmation du système IBM 360 par une équipe de "Computer Usage
Company" sous la direction de A. Opler (Dunod Paris 1967).

Principe de programmation des ordinateurs
par A. Lauret (Masson).

Manuel de référence CAE SYMBOL - METASYMBOL (CII série CAE 90 -
Réf. C90 05 06 E/Fr - Mars 1967).

LISTE DES ANNEXES

- 1 - Tableau des caractères de la flexowriter
- 2 - Instructions CAB 500 - Liste alphabétique (1 et 2)
- 3 - Liste des instructions par classes
- 4 - Indice de saut généralisé
- 5 - Place du digit de décalage en registre ou en mémoire
- 6 - Utilisation des instructions
- 7 - Diagnostics des erreurs détectées à l'assemblage
- 8 - Codes d'instruction binaires
- 9 - Tables des codes opération
- 10 - Essais du système
- 11 - Utilisation des registres

Directives : RES, FIN, TEXTE, DFC

Organigrammes : SP1, SP2, SP3, SP4, SP5, SP6, SP7, SP8

Sous-programmes : écriture DCB,
lecture et écriture hexadécimale

Listes d'assemblage : TEST1, TEST2, TEST3, TEST4, TEST5, TEST6

Listes de chargement

CODE INTERNE
DECIMAL

IMPRESSION			
	HAUT	PERFO	BAS
0	0	• •	EXP 0
1	1	•• • •	EXP 1
2	2	•• • •	EXP 2
3	3	•• • •	EXP 3
4	4	•• ••	EXP 4
5	5	• •••	EXP 5
6	6	• •••	EXP 6
7	7	•• ••••	EXP 7
8	8	•• ••	EXP 8
9	9	• •• •	EXP 9
10	+	•• ••	.
11	-	•• ••••	IND 0
12	,	• •••	IND 7)
13	x	•• ••••	
14	=	•• ••••	IND 9
15	ESPACE	• •••••	
16	/	••• •	(
17	Δ	•• • •	IND 8
23	BAS	•• ••••	
24	HAUT	••••	
25	R C	••••• •	
26	ESP.ARR.	••••• •	
30	TAB	••••••	
31	EFFACEMENT	••••••••	

CODE INTERNE
DECIMAL

IMPRESSION			
	HAUT	PERFO	BAS
32	AVANCE B.	• •	
33	A	• • •	'
34	B	• • •	IND 5
35	C	• • ••	IND 3
36	D	• • •	<
37	E	• •••	#
38	F	• •••	-?
39	G	•••••	<
40	H	••	
41	I	• •• •	IND 1
42	J	• •• •	IND j
43	K	• • ••	IND k
44	L	• •••	8
45	M	•••••	>
46	N	•••••	IND 6
47	O	• •••••	:
48	P	• •	π
49	Q	•• •• •	*
50	R	• • • •	
51	S	• • • •	Σ
52	T	• • • •	>
53	U	• • • •	1
54	V	• •••	IND 4
55	W	• •••••	IND 1
56	X	• •••	IND 2√
57	Y	••• •	"
58	Z	••• •	

ANNEXE 1 - TABLEAU DES CARACTERES DE LA FLEXOWRITER

AUTOCODE	MNEMONIQUE			SIGNIFICATION
	BINAIRE		M.P.	
A	9 8	A	C	ADDITION (VIRGULE FIXE)
AL	A C	AL	MP	ADDITION DOUBLE LONGUEUR (NORMALISEE)
AF	6 C	AN	MP	ADDITION NORMALISEE (SIMPLE LONGUEUR)
ARC	D C	ARC	MP	ARC COSINUS
ARS	D 4	ARS	MP	ARC SINUS
ART	C 4	ART	MP	ARC TANGENTE
R	C 8	BC	C	BRANCHEMENT CONDITIONNEL
RC	4 8	BRC	C	BRANCHEMENT ET RETOUR CONDITIONNEL
A... E	9 A	C	C	CHARGEMENT (VIRGULE FIXE)
S... E	B A	CC	C	CHARGEMENT COMPLEMENTE
SL... E	B 6	CCL	MP	CHARGEMENT COMPLEMENT LONG (D.L.)
SF... E	7 6	CCN	MP	CHARGEMENT COMPLEMENT NORMALISE (S.L.)
AL... E	A E	CL	MP	CHARGEMENT LONG (D.L.)
AF... E	6 E	CN	MP	CHARGEMENT NORMALISE (S.L.)
COS	3 4	COS	MP	COSINUS
D... Q	E 2	DLQ	C	DECALAGE A DROITE LONG (R14)
D... T	E 0	DLT	C	DECALAGE A DROITE LONG (R13)
C... Q	C 2	DQ	C	DECALAGE A DROITE (R14)
LC...	9 4	DSL	MP	CONVERSION DOUBLE-SIMPLE LONGUEUR
C... T	C 0	DT	C	DECALAGE A DROITE (R13)
Q... E	0 E	DVE	MP	DIVISION ENTIERE
Q	0 C	DVF	MP	DIVISION FRACTIONNAIRE
QL	8 C	DVL	MP	DIVISION LONGUE (D.L.)
QF	4 C	DVN	MP	DIVISION NORMALISEE (S.L.)
E	F 8	EA	C	ECRITURE ALPHANUMERIQUE
I... Q	7 A	ETQ	C	INTERSECTION (ET) (R14)
I... T	7 8	ETT	C	INTERSECTION (ET) (R13)
EXP	B C	EXP	MP	EXPONENTIELLE
NF	3 E	FN	MP	CONVERSION VIRGULE FIXE-VIRGULE NORMALISEE (S.L.)
G... Q	F 2	GLQ	C	DECALAGE A GAUCHE LONG (R14)
G... T	F 0	GLT	C	DECALAGE A GAUCHE LONG (R13)
K... Q	D 2	GQ	C	DECALAGE A GAUCHE (R14)
K... T	D 0	GT	C	DECALAGE A GAUCHE (R13)
F	3 8	HSE	C	HALTE ET SEQUENCE

AUTOCODE	MNEMONIQUE		M.P.	SIGNIFICATION
	BINAIRE			
IL..E..F	9 E	ILN	MP	IMPRESSION D'UN NOMBRE LONG (D.L.) FORME NORMALISEE
IA	2 E	IMA	MP	IMPRESSION D'ADRESSE
IE	1 C	IME	MP	IMPRESSION D'UN NOMBRE ENTIER
IV	1 E	IMF	MP	IMPRESSION D'UN NOMBRE FRACTIONNAIRE
IL...D	9 C	IML	MP	IMPRESSION D'UN NOMBRE LONG (D.L.)
IF	5 C	IMN	MP	IMPRESSION D'UN NOMBRE NORMALISE (S.L.)
IA...E	2 C	IMX	MP	IMPRESSION D'INDICE
A..ELO	9 A	LA	C	LECTURE ALPHANUMERIQUE
LGD	7 C	LGD	MP	LOGARITHME DECIMAL
LGN	7 E	LGN	MP	LOGARITHME NEPERIEN
LL	A 4	LL	MP	LECTURE D'UN NOMBRE DECIMAL ET REPRESENTATION D.L.
LF	6 4	LN	MP	LECTURE D'UN NOMBRE DECIMAL ET REPRESENTATION S.L.
M	0 4	M	MP	MULTIPLICATION (FIXE)
M...S	0 6	MC	MP	MULTIPLICATION COMPLEMENTEE (FIXE)
ML	8 4	ML	MP	MULTIPLICATION LONGUE (D.L.)
ML...S	8 6	MLC	MP	MULTIPLICATION LONGUE COMPLEMENTEE
MF	4 4	MN	MP	MULTIPLICATION NORMALISEE (S.L.)
MF...S	4 6	MNC	MP	MULTIPLICATION NORMALISEE COMPLEMENTEE
CF	3 C	NF	MP	CONVERSION VIRGULE NORMALISEE (S.L.) FIXE
X	0 0	NOP	C	NON OPERATION
V...Q	5 A	OEQ	C	DISJONCTION (OU EXCLUSIF) (R14)
V...T	5 8	OET	C	DISJONCTION (OU EXCLUSIF) (R13)
YE	1 6	RE	MP	RACINE CARREE ENTIERE
YV	1 4	RF	MP	RACINE CARREE FRACTIONNAIRE
YL	8 E	PL	MP	RACINE CARREE LONGUE (D.L.)
YF	5 4	RN	MP	RACINE CARREE NORMALISEE (S.L.)
S	B 8	S	C	SOUSTRACTION (FIXE)
CL	9 6	SDL	MP	CONVERSION SIMPLE - DOUBLE LONGUEUR
SIN	3 6	SIN	MP	SINUS
SL	B 4	SL	MP	SOUSTRACTION LONGUE
SF	7 4	SN	MP	SOUSTRACTION NORMALISEE (S.L.)
T	1 8	T	C	TRANSFERT
TG	C C	TG	MP	TANGENTE
T...E	1 A	TRZ	C	TRANSFERT ET REMISE A ZERO

CLASSE	ADRESSE1	PROPRIETES PARTICULIERES						INSTRUCTIONS (MNEMO)
		OP1		OP2		OP3		
		LIMITES	CADRAGE	LIMITES	CADRAGE	LIMITES	CADRAGE	
1	3	≤15	B12	≤ 15 ≠OP1	B22			A-C-CC-ETQ ETT-OEQ-OET-T-TRZ S
		≤15 ≠ACL	B12	≤ 15 ≠OP1 ≠ACL	B22			DIQ-DLT-DQ-DT CLQ-GLT-GQ-GT
2	2	≤ 8	B12	≤ 8	B22			AN-CCN-CN-DVE- DVF-DVN-FN-M-MC- MN-MNC-NF-RE-RF RN-SN
3	2	≤ 8	B12	≤ 15	B22			IMA-IME-IMF-IMI
4	2	≤ 7	B5	≤ 7	B22			AL-CCL-CL-DVL-ML MLC-SL
		≤ 7	B12	≤ 7	B22			DSL-RL-SDL
5	2	≤ 7 ≠0	B12	≤ 7 ≠0	B22			ARC-ARS-ART-COS EXP-LGD-LGN-SIN
6	3	≤15	B12	≤ 7	B9	≤ 15	B5	BC-BRC
7	1	≤15	B22					LA
8	2	≤ 8	B22					LN
9	2	≤ 7	B5					LL
10	3	≤15	B12					EA
11	1	≤ 8	B22	≤ 31	B5	≤ 31	B12	IMN
12	1	≤ 7	B22	≤ 31	B5	≤ 31	B12	IML
13	1	≤ 7	B22	≤ 31	B12			ILN

* - Adresse 1 - 3:Registre, Mémoire, Immédiat - 2:Registre, Mémoire
1:Registre

ANNEXE 3 - LISTE DES INSTRUCTIONS PAR CLASSES

L'indice de saut porte sur une instruction I , opérande 1 en mémoire N

N' : Numéro dans la piste, de la mémoire citée dans l'instruction

P': " " " " " " " contenant l'instr. I+1

(prendre la valeur P' la plus proche de N)

n'	p'		n'	p'		n'	p'		n'	p'	
0	1	66	40	41	106	80	81	17	120	121	57
1	2	67	41	42	107	81	82	18	121	122	58
2	3	68	42	43	108	82	83	19	122	123	59
3	4	69	43	44	109	83	84	20	123	124	60
4	5	70	44	45	110	84	85	21	124	125	61
5	6	71	45	46	111	85	86	22	125	126	62
6	7	72	46	47	112	86	87	23	126	127	63
7	8	73	47	48	113	87	88	24	127	0	64
8	9	74	48	49	114	88	89	25			
9	10	75	49	50	115	89	90	26			
10	11	76	50	51	116	90	91	27			
11	12	77	51	52	117	91	92	28			
12	13	78	52	53	118	92	93	29			
13	14	79	53	54	119	93	94	30			
14	15	80	54	55	120	94	95	31			
15	16	81	55	56	121	95	96	32			
16	17	82	56	57	122	96	97	33			
17	18	83	57	58	123	97	98	34			
18	19	84	58	59	124	98	99	35			
19	20	85	59	60	125	99	100	36			
20	21	86	60	61	126	100	101	37			
21	22	87	61	62	127	101	102	38			
22	23	88	62	63	0	102	103	39			
23	24	89	63	64	0	103	104	40			
24	25	90	64	65	1	104	105	41			
25	26	91	65	66	2	105	106	42			
26	27	92	66	67	3	106	107	43			
27	28	93	67	68	4	107	108	44			
28	29	94	68	69	5	108	109	45			
29	30	95	69	70	6	109	110	46			
30	31	96	70	71	7	110	111	47			
31	32	97	71	72	8	111	112	48			
32	33	98	72	73	9	112	113	49			
33	34	99	73	74	10	113	114	50			
34	35	100	74	75	11	114	115	51			
35	36	101	75	76	12	115	116	52			
36	37	102	76	77	13	116	117	53			
37	38	103	77	78	14	117	118	54			
38	39	104	78	79	15	118	119	55			
39	40	105	79	80	16	119	120	56			

Nb de positions de décalage	Décalage à droite		Décalage à gauche	
	Place du bit de décalage	Évaluation du bit de décalage	Place du bit de décalage	Évaluation du bit de décalage
0	b	32	b	1
1	b	16	b	2
2	b	8	b	3
3	b	4	b	4
4	b	2	b	5
5	b	1	b	6
6	b		b	7
7	b		b	8
8	b		b	9
9	b		b	10
10	b		b	11
11	b		b	12
12	b		b	13
13	b		b	14
14	b		b	15
15	b		b	16
16	b		b	17
17	b		b	18
18	b		b	19
19	b		b	20
20	b		b	21
21	b		b	22
22	b		b	23
23	b		b	24
24	b		b	25
25	b		b	26
26	b		b	27
27	b		b	28
28	b		b	29
29	b		b	30
30	b		b	31
31	b		b	32
32	b		b	33

* Si, dans l'instruction, on veut programmer un décalage immédiat, c'est le nombre de positions de décalage qui doit être écrit et non l'évaluation donnée ici.

OPERATIONS	POURCENTAGE
Addition et chargement en virgule fixée	31,2
Transfert de commande	12,1
Transfert de données	10,4
Ecriture alphanumérique	9,7
Soustraction et chargement complémenté en virgule fixée	8,4
Dijonction	4,8
Intersection	2,2
Multiplication virgule normalisée	1,8
Décalage court à droite	1,6
Addition virgule normalisée	1,5
Soustraction virgule normalisée	1,3
Décalage à droite (long)	1,2
Décalage à gauche (court)	1
Division virgule normalisée	1
Impression virgule normalisée	1
Impression entière	0,5
Racine carrée virgule normalisée	0,5
Lecture en virgule normalisée	0,5
Décalage à gauche (long)	0,5
Multiplication en virgule fixée	0,4
Impression d'adresse et d'indice	0,3
Conversions	0,2
Division virgule fixée	0,1
Racine carrée virgule fixée	0,1
Non analysés	7,7
Les statistiques ci-dessus ont été établies sur différents programmes scientifiques représentant 20 000 instructions élémentaires.	
OPERANDE 1	POURCENTAGE
Registre	38,70
Mémoire	20,23
Immédiat V	18,32
Immédiat V conditionnel (branchement)	8,86
Immédiat W	4,80
Alphanumérique (lecture)	4,16
Mémoire conditionnelle (branchement)	3,00
Non analysée	1,93
Statistiques établies sur programmes représentant 30 000 instructions élémentaires.	

CLASSE 0 : Arrêt de l'assemblage	
01	Dépassement de capacité à la lecture du programme
02	Dépassement de capacité de la table des symboles
03	Dépassement de capacité de la table des instructions binaires
04	Dépassement de capacité de la table des erreurs
CLASSE 1 : Sans influence sur l'assemblage de l'instruction qui suit	
10	Etiquette interdite
11	Etiquette illégale (caractères spéciaux
12	Multidéfini-tion d'un symbole interne
13	Multidéfini-tion d'un symbole externe
CLASSE 2 : Non assemblage de l'instruction erronée	
20	Code opération illégal (inconnu, format non respecté)
21	Code 1er opérande illégal (non compatible avec instruction)
22	Défini-tion opérande registre illégale
23	Défini-tion opérande immédiat illégale
24	Défini-tion symbole illégale
25	Expression symbolique illégale (externe
26	Expression symbolique illégale (translatabilité)
27	Expression symbolique illégale (valeur maximale)
28	Défini-tion 2ème opérande illégale
29	Défini-tion 3ème opérande ou fin d'instruction illégale
30	Programme sans directive "FIN"

CODE		SIGNIFICATION	OBSERVATIONS
DECIMAL	BINAIRE		
0	0000	Adresse de chaînage	Chargeur
1	000 <u>1</u>	Adresse de chaînage et diagnostic d'erreur Δ	Liste d'assemblage
2	0010	Adresse non translatable *	Chargeur
3	001 <u>1</u>	Adresse non translatable et diagnostic d'erreur	Liste d'assemblage
4	0100	Adresse translatable	Chargeur
5	010 <u>1</u>	Adresse translatable et diagnostic d'erreur	Liste d'assemblage
6	0110	Constante (simple ou double longueur * sans facteur multiplicatif)	Liste et chargeur
7	011 <u>1</u>	Constante et diagnostic d'erreur	Liste d'assemblage
8	1000	Chargement de constante multiple ou réservation	Liste et chargeur
9	100 <u>1</u>	Chargement de constante et erreur	Liste d'assemblage
10	1010	Arrêt impression programme objet	Liste d'assemblage
11	1011	Début bande programme objet 1er mot suivant : taille du programme 2è mot suivant : nom symbolique du programme	Gestion du chargeur
12	1100	Fin de programme	Liste d'assemblage et gestion du chargeur
13	1101	Fin de chargement (1ère phase)	Gestion chargeur
14	1110	Définition externe	Chargeur
15	1111	Référence externe	Chargeur

*. du point de vue chargement, ces codes (2 et 6) pourraient être confondus; cette distinction n'est utile que pour la liste d'assemblage.

Δ . pour les codes 0 à 9; le digit B1 représente le drapeau d'erreur.

Table des MATIERES

Table des CODES OPERATION

A 038506FE
B 0387070A
C 038E070E
D 0397071C
E 039B0730
F 039E0738
G 03A2073C
H 03A30744
I 03AA0746
J 00000000
K 00000000
L 03AF0754
M 03B5075E
N 03B7076A
O 03B9076E
P 00000000
Q 00000000
R 03BE0772
S 03C3077C
T 03C70786

A 00000000
98010037
AN 0000002E
6C00802A
ART 00000CB4
C4008023
ARS 00000CB3
D4008023
ARC 00000CA3
DC008023
AL 0000002C
AC008025
BC 00000023
C801023E
BRC 00000CA3
4801023E
C 00000000
9A010037
CN 0000002E
6E00802A
COS 00000BF3
34008023
CC 00000023
BA010037
CCN 000008EE
7600802A
CL 0000002C
AE008025
CCL 000008EC
B6008025
DT 00000034
C0010034
DQ 00000031
C2010032
DLT 00000B34
E0010034
DLQ 00000B31
E2010032
DFC 000009A3
00018825
DVN 00000DAE
4C00802A
DVE 00000DA5
0E00802A
DVF 00000DA6
0C00802A
DVL 00000DAC
8C008025
DSL 00000CEC
940082BE

EA 00000021
F8010287
ETT 00000D34
78010037
ETQ 00000D31
7A010037
EXP 00000E30
BC008023
FIN 00000A6E
000187D0
FN 0000002E
3E00802A
GT 00000034
D001002F
GQ 00000031
D201002D
GLT 00000B34
F001002F
GLQ 00000B31
F201002D
HSE 00000CE5
38018000
IMN 00000B6E
5C000299
IME 00000B65
1C008028
IMF 00000B66
1E008028
IMX 00000B78
2C008028
IMA 00000B61
2E008028
IML 00000B6C
9C00029B
ILN 00000B2E
9E00029E
LA 00000021
9A000269
LN 0000002E
6400826C
LGD 000009E4
7C008023
LGN 000009EE
7E008023
LL 0000002C
A4008272
MN 0000002E
4400802A
M 00000000
0400802A

ML 0000002C
84008025
MNC 00000BA3
4600802A
MC 00000023
0600802A
MLC 00000B23
86008025
NOP 00000BF0
00018000
NF 00000026
3C00802A
OET 00000974
58010037
OEQ 00000971
5A010037
RES 00000973
0001878F
RN 0000002E
5400802A
RE 00000025
16008024
RF 00000026
14008024
RL 0000002C
8E0082BE
S 00000000
B8010037
SN 0000002E
7400802A
SIN 00000A6E
36008023
SL 0000002C
B4008025
SDL 0000092C
960082BE
T 00000000
18008037
TRZ 00000CBA
1A008037
TEXTE 00978D25
000187A3
TG 00000027
CC008023

FIN de BANDE	38000000
-----------------	----------

	PROGRAMME	PTEST1	PTEST2	PTEST3	PTEST4
LANGAGE ASSEMBLAGE	Nombre de lignes	43	80	22	42
	Nombre d'instructions ou constantes générées	38	68	104	35
	Nombre de mots programme-source	170	610	86	278
	Temps d'assemblage (s)	45	120	25	65
	Temps d'édition liste d'assemblage (s)	260	840	220	345
	Temps d'édition liste binaire (s)	60	145	20	
	Temps de chargement (s)	10	12	8	
	Temps d'édition liste de chargement (s)	40	30		
LANGAGE AUTOCODE	Nombre de lignes ou instructions	38	68		
	Temps de chargement en autocode (s)	15	20		
	Temps d'édition liste binaire (s)	8	15		
	Temps de chargement en code "S" (s)	48	60		
	Temps d'édition liste binaire S (s)	30	55		

N°	P0	P1	P2 - P3		P4			CHA - P2		
0	RP	RP	RP		RT	RT1	KER	RT	REXT	
1	K1	ROP	ROP	RN	RSYMB	RT2	RG	KL		
2	K2	KA	KA		PTR2		KA	PTR2		
3	RT	RL	RL		PTR3		KL	PTR3		
4		RΔ	RMAXI	RSG	Garage R12		PTR4		RB	RADE
5		RS	RTI	RDL	RCT	RN	KN	Garage R12		KP
6		RI	RI	RB	RCP	RI	Garage R12		RI	PTR1
7		RG	RCI	ROP1	RAM			REXE		
8		RAD	RAM	RC	RAD		RB	RAD	RAD	RN
9		RET	REXT	KM	ROP2		RT		RATM	RT2
10		Garage RX	RTR		RLIST			RT'		
11		PTRS	PTRS		RAM	RAD	Garage R12		RMSQ	RT1
12		Garage CO	Garage CO		Garage CO			Garage CO		
13	ACL1	ACL1	ACL1		ACL1			ACL1		
14	ACL2	ACL2	ACL2		ACL2			ACL2		
15	RX	RX	RX		RX			RX		

ANNEXE 11 - UTILISATION DES REGISTRES GENERAUX

ASSEMBLEUR : DIRECTIVE " RES "

0	RV(ΔY14	A (ER20)	(93Δ P3)
	V3QV5		
	T13ROY14		
	COTV25Y14		
	A3V1Y14		
	RV1(ΔY14		
	A15V1		
	A(EM)+		
	COTV24		
	A3EV1		
10	TOR14	TEST CHIFFRE	
	S14V10		
	RV(ΔP14	A (ER23)	(99Δ P3)
	RCV(Δ	A (SP DECIMAL)	(138Δ P2)
	RV(ΔZ1	A (ER23)	
	A10EV8	RTR=8	
	A5EV1	RSL=1	
	RV(Δ	A (AO7 - SP FIN D'INSTRUCTION)	(461Δ P2)
	F		
	FE		

ASSEMBLEUR : DIRECTIVE " FIN "

0	RCV(ΔY9 T15R10 A15EM(Δ X V15QR11 RV11ΔZ14 A9EM1+ I9QV32767 RV11ΔY14 A7EM(Δ+ RV13Δ	A (ER10) A (TABSB) RESERVE NOM SYMBOLIQUE → ROP1	(48 Δ P3) (31 Δ P1)
11	A7EV62415 A7W1950	5 ESPACES → ROP1	
13	T10ER15 V0QV25 RV29ΔZ14 V0QV30 RV(ΔY14 V3QV5 T13ROY14 COTV25Y14 A3V1Y14 RV27ΔY14 A15V1 A(ΔEM(Δ+ COTV24 A3EV1 V0QV30 RV31ΔY14	TEST RC TEST TAB A (ER20) CARACTERE SUIVANT TEST TAB	(93 Δ P3)
27	RV31ΔY14		
29	A6EW49152 RV73Δ	B32 ET B33 → (RI)	
31	TOR14 S14V10 RV36ΔP14 A12EV53Δ RV(Δ	TEST CHIFFRE A (SP DECIMAL)	(138 Δ P2)
36	S14V23 RV(ΔN14 TOR1 V3QV5 T13ROY14 COTV25Y14 A3V1Y14 RV48ΔY14 A15V1 A(ΔEM(Δ+ COTV24 A3EV1 V1QV56 RV56ΔY14	A (ER24) CARACTERE SUIVANT TEST "X"	(101 Δ P3)
48			

	V0QV14	TEST "="	
	RV56AY14		
53	RCV(Δ)	A (SP HEXADECIMAL)	(216 Δ P2)
	T1ER6	(ROP) → (RI)	
	A6W1	B18 → RI	
	RV73Δ		
56	RCV(Δ)	A (SP LECTURE SYMBOLE -C01)	(135 Δ P1)
	RCV(Δ)	A (SP CADRAGE SYMBOLE)	(400 Δ P2)
	T15ER10		
	A15EM(Δ)	A (TABSB)	(31 Δ P1)
	V15QR11		
	T10ER15Z14		
63	RV(ΔZ14	A (ER25)	(103 Δ P2)
	V1QM)+	RECHERCHE SYMBOLE	
	RV67ΔZ14		
	A15V2		
	RV63Δ		
67	A6EM1+	MISE EN PLACE ADRESSE EXECUTION → RI	
	T10R15		
	I6QW5	TEST B18-B20	
	RV(ΔY14	A (ER25)	(103 Δ P2)
	I6QV32767		
	T14R6		
73	A15V2		
	T15R14		
	S14M(Δ)	A (DERNIERE ADRESSE TABPR)	(537 Δ P2)
	FN14	ARRET (FIN MACRO)	
	A15EM(Δ)	A (TABIN)	
	T7EM1+	ROP1	
	T2EM)+	(KA) DONNE LE NOMBRE D'INSTRUCTIONS GENEREES	
	A10EV12	RTR=12	
	A12EV(Δ)	A (LISTE)	(C Δ P4)
	RV(Δ)	A (SP FIN INSTRUCTION - B07)	(462 Δ P2)
	F	RESERVE	
	F		
	FE	FIN DE BANDE	

ASSEMBLEUR : DIRECTIVE " TEXTE "

0	RV0ΔY14	A (ER20)	(48 Δ P3)
	A5EV1	1 → (RSL)	
	A9EV0	0 → (KM)	
	A10EV6	6 → (RTR)	
	A6EV0	0 → (RI)	
5	A8EV0	0 → (RC)	
	A9V1	EVOLUTION (KM)	
7	V3QV5	LECTURE CARACTERE	
	T13ROY14		
	C0TV25Y14		
	A3V1Y14		
	RV16ΔY14		
	A15V1		
	A0EM0+		
	C0TV24		
	A3EV1		
16	V0QV25	TEST "RC"	
	RV26ΔZ14		
	K6QV6	MISE EN PLACE CARACTERE	
	A14R0		
	T14R6		
	A8V1		
	V8QV5		
	RV7ΔY14		
	A12EV5Δ	A (RETOUR) → R12	
	RV(Δ	A (DO7)	(472 Δ P2)
26	A0EV4	4- (RC) → (RP)	
	S0R8		
28	RV34ΔN0	MOT COMPLETE PAR DES ESPACES (15)	
	K6QV6		
	A14V15		
	T14R6		
	S0V1		
	RV28Δ		
34	V9QV1	TEST (KM)	
	RV39ΔZ14		
	A0EW49152	MISE EN PLACE NOMBRE D'INSTRUCTIONS	
	A0R9	GENEREES DANS LE MOT DE SEPARATION	
	T0M1+	DU PROGRAMME SOURCE	
39	A12EV(Δ	A (RETOUR) → R12 (W2)	(534 Δ P2)
	RV(Δ	A (DO7)	
	F	RESERVE	
	F		
	F		
	F		
	FE	FIN DE BANDE	

ASSEMBLEUR : DIRECTIVE " DFC "

<u>0</u>	FY14 V3QV5 T13ROY14 COTV25Y14 A3V1Y14 RV10AY14 A15V1 AOEMO+ COTV24 A3EV1 TOR14 S14V10 A4EV1P14 RV17AP14 F FZ1 T1ER4 T4M18AS X V3QV5 T13R1Y14 C1TV25Y14 A3V1Y14 RV28AY14 A15V1 A1EMO+ C1TV24 A3EV1 V1QV14 FY14 V0QV38 RV34AY14 A10EV55Δ RV41Δ V0QV56 RV38AY14 A10EV265Δ RV41Δ V0QV51 RV43AY14 A10EV280Δ T4R14 RV47Δ V0QV36 FY14 A10EV671Δ K4QV1 A14R12 X S14M50AS	<u>50</u>	X FP14 T11EM53AS X RR10 V3QV5 T13ROY14 COTV25Y14 A3V1Y14 RV64AY14 A15V1 AOEMO+ COTV24 A3EV1 S4EV1 V0QV10 RV70AZ14 V0QV11 RV79AY14 A4EV0 V3QV5 T13ROY14 COTV25Y14 A3V1Y14 RV79AY14 A15V1 AOEMO+ COTV24 A3EV1 TOR14 S14V10 RV275AP14 F T1R7 V0QV12 RV88AZ14 RV154AB7 RV277Δ A8EV8 A9EV10 V3QV5 T13ROY14 COTV25Y14 A3V1Y14 RV99AY14 A15V1 AOEMO+ COTV24 A3EV1 V3QV5	<u>100</u>	T13R1Y14 C1TV25Y14 A3V1Y14 RV108AY14 A15V1 A1EMO+ C1TV24 A3EV1 T1R14 S14V10 RV133AP14 K0QV3 A1R14 A1R0 A0R1 K9QV3 A14R9 A9R14 X V3QV5 T13R1Y14 C1TV25Y14 A3V1Y14 RV128AY14 A15V1 A1EMO+ C1TV24 A3EV1 T1R14 S14V10 RV133AP14 S8V1 RV111AY8 T13ER5 T9ER13 Q0FR13 RV151AY8 S1V5 A0V1P1 V3QV5 T13R1Y14 C1TV25Y14 A3V1Y14 RV148AY14 A15V1 A1EMO+ C1TV24 A3EV1 T1R14 S14V10	<u>150</u>	RV139AN14 T5ER13 TOR5 T1ER0 V0QV16 RV222AY14 V3QV5 T13ROY14 COTV25Y14 A3V1Y14 RV165AY14 A15V1 AOEMO+ COTV24 A3EV1 S11EV1 V0QV10 RV171AZ14 V0QV11 RV180AY14 A11EV0 V3QV5 T13ROY14 COTV25Y14 A3V1Y14 RV180AY14 A15V1 AOEMO+ COTV24 A3EV1 TOR14 S14V10 RV277AP14 F T1R14 S14V33 RV277AP14 RV222AZ1 T15ER10 RV209AP11 A15ER1 A15V1 T15R14 S14V18 RV200AN14 S15V17 A1EMO+ K1QV17 T14R1 RV201Δ
----------	---	-----------	---	------------	--	------------	---

200 A1EMO+
I7QV1
A5W16384Y14
I7QV2
A5W32768Y14
T5R14
G7QR1
T14R7
RV221Δ
A15EV33
S15R1
T15R14
S14V17
RV219ΔN14
S15V16
A1EMO+
K1QV16
T14R1
RV220Δ
A1EMO+
C7QR1
T10ER15
S1ER7P4
T1R7P4
A5EV1
A1EM18Δ
V1QV1
RV233ΔY14
A10EV6
T7R6
T9R7N5
A1V1N5
RV241Δ
A10EV8
T1R6
RV238ΔN5
A6V32768
RV241Δ
A6V65536
K1QV1
T14R1
A11EM53Δ
F
X
X
X
X
X
X
X
X

250 X
X
X
X
X
X
X
X
X
X
X
X
X
S1EV1
F
RV277ΔN1
S8V9
RV277ΔP8
A5EVO
T1R7
RV154Δ
F
F
A11EM53Δ
F
A11EM53Δ
F
F
V3QV5
T13R1Y14
C1TV25Y14
A3V1Y14
RV289ΔY14
A15V1
A1EMO+
C1TV24
A3EV1
T1R4
S4V10
RV304ΔN4
S4V1
RV295ΔZ4
RV401ΔP4
V3QV5
T13R1Y14
C1TV25Y14
A3V1Y14
RV304ΔY14

300 A15V1
A1EMO+
C1TV24
A3EV1
V3QV5
T13R0Y14
C0TV25Y14
A3V1Y14
RV313ΔY14
A15V1
A0EMO+
C0TV24
A3EV1
V0QV12
A7EV1Z14
RV637ΔZ14
A14ERO
S14V10
A7EVO
RV342ΔP14
K1QV1
AOR14
K1QV3
AOR14
TOR1
I0QW31744
RV304ΔZ14
V3QV5
T13R0Y14
C0TV25Y14
A3V1Y14
RV336ΔY14
A15V1
A0EMO+
C0TV24
A3EV1
V0QV12
RV388ΔZ14
S0V10
A7V1N0
RV327ΔN0
A0V10
V0QV17
RV408ΔY14
S11EV1
V3QV5
T13R0Y14
C0TV25Y14
A3V1Y14
RV354ΔY14

350 A15V1
A0EMO+
C0TV24
A3EV1
TOR9
V0QV11
A11EV0Z14
RV345ΔZ14
V0QV10
RV345ΔZ14
V3QV5
T13R0Y14
C0TV25Y14
A3V1Y14
RV369ΔY14
A15V1
A0EMO+
C0TV24
A3EV1
TOR14
S14V10
RV385ΔP14
K9QV3
AOR14
AOR9
A9R0
V3QV5
T13R0Y14
C0TV25Y14
A3V1Y14
RV385Y14
A15V1
A0EMO+
C0TV24
A3EV1
A7R9N11
S7R9Z11
RV408Δ
V3QV5
T13R0Y14
C0TV25Y14
A3V1Y14
RV397ΔY14
A15V1
A0EMO+
C0TV24
A3EV1
S0V10
RV388ΔN0
RV341Δ

[illegible]

450

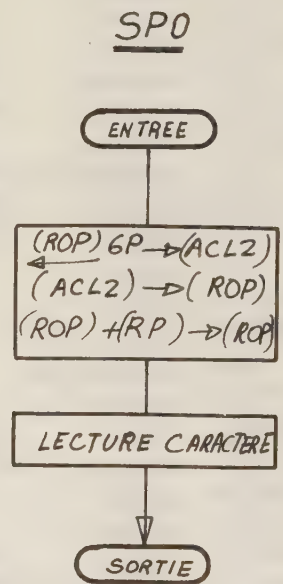
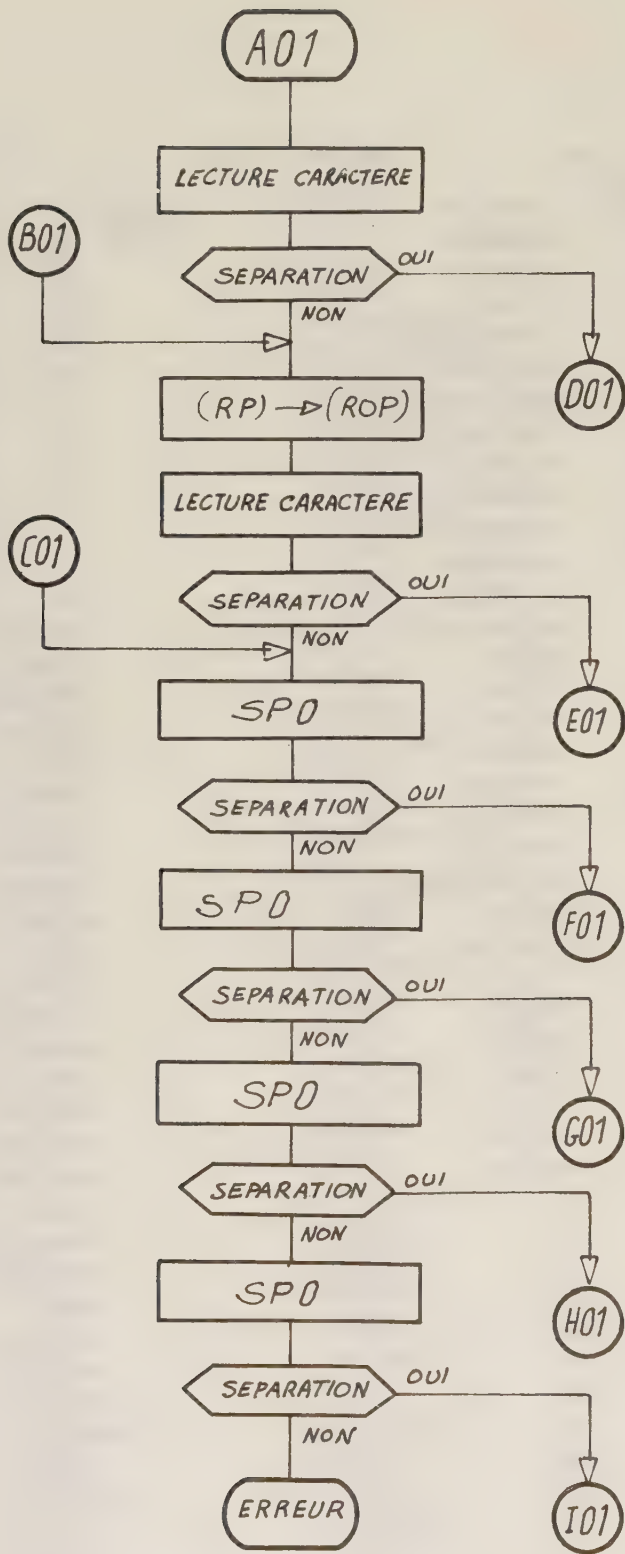
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
A8V1Y1
D1QR10Y1
T14R1
S1W8192
RV481ΔY1
RV481ΔN4
S1ER14
S8V1
K1QV1
A1ER14
RV483Δ
S1ER14Z4
A1ER14N4
I1QV127
S1R14
T15ER12
A11ER7P7
S11ERTN7
S11V20
RV40XΔP11
T7ER15
A15EM611Δ+
K15QV26
T14ER11
C11QV26
A14ER11P11
S14ER11N11
S14V64
RV40XΔP14
I15QV128

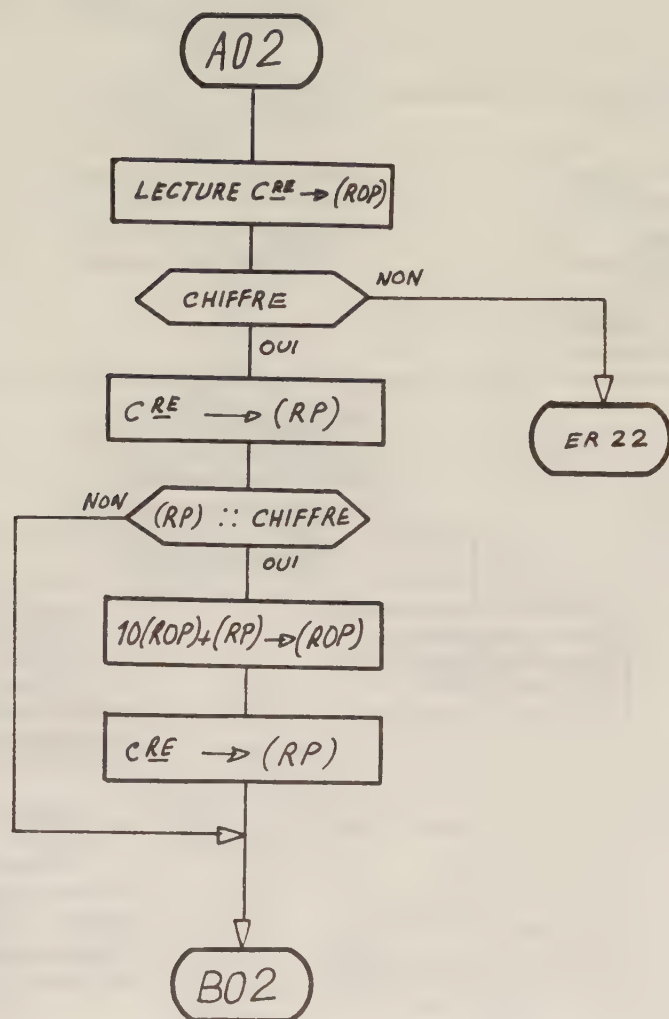
500 A7R1Y14
C7QV1
I15QV256
A7R1Y14
C7QV1
I15QV512
A7R1Y14
C7QV1
I15QV1024
A7R1Y14
C7QV1
I15QV2048
A7R1Y14
C7QV1
I15QV4096
A7R1Y14
C7QV1
I15QV8192
A7R1Y14
C7QV1
I15QV16384
A7R1Y14
C7QV1
I15QV32768
A7R1Y14
C7QV1
I15QV65536
A7R1Y14
C7QV1
I15QW1
A7R1Y14
C7QV1
I15QW2
A7R1Y14
C7QV1
I15QW4
A7R1Y14
C7QV1
I15QW8
A7R1Y14
C7QV1
I15QW16
A7R1Y14
C7QV1
I15QW32
A7R1Y14
C7QV1
I15QW64
A7R1Y14
C7QV1

550 I15QW128
A7R1Y14
C7QV1
I15QW256
A7R1Y14
C7QV1
I15QW512
A7R1Y14
C7QV1
I15QW1024
A7R1Y14
C7QV1
I15QW2048
A7R1Y14
C7QV1
I15QW4096
A7R1Y14
C7QV1
I15QW9192
A7R1Y14
C7QV1
K7QV1
RV575ΔD14
T14ER7
S8V1
A7V64
RV579ΔB7
C7QV1
A8V1
T13ER9
I7TV127
S7R13
I15TV127
K13QV26
C14TV26
A8R14
I8TV127
A7R13Y7
T9ER13
T12ER15
RV224Δ
X
X
X
X
X
X
X
X
X
X

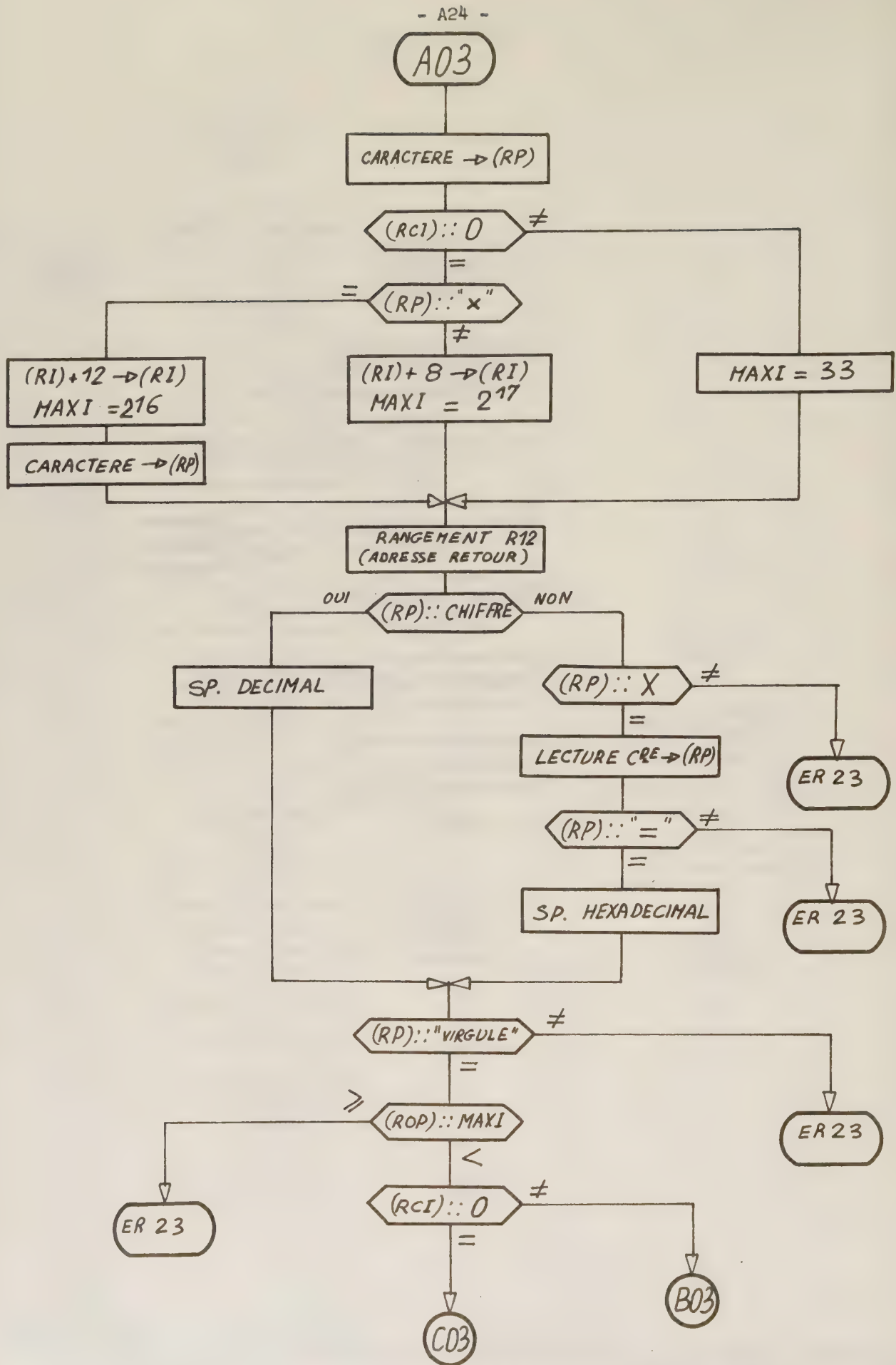
600	X	650	K1QV1	700	T2ER12	750	RV759AY14
	X		AOR14		T14ER2		I13QV4032
	X		K1QV3		A9EV0		RV756AY14
	X		AOR14		S8EV1		I13QV63
	X		TOR1		V3QV5		A11W252
	X		I0QW31744		RV709AZ14		S5V6
	X		RV637AZ14		C2QV25		A11W3
	X		V3QV5		A3V1		A11V114688
	X		T13ROY14		RV713A		S5V5
	X		COTV25Y14		A15V1		A11V16256
	X		A3V1Y14		A2EM0+		S5V7
	X		RV666AY14		C2QV24		A11V127
	X		A15V1		A3EV1		S5V7
	X		A(EM)0+		T2R10		T12ER14
	X		COTV24		T2ER0		G13QR11
	X		A3EV1		T14ER2		G14TV1
	X		A14ER0		V10QV12		RV783AD13
	X		S14V10		A8EV0Z14		S5V1
	X		RV657AN14		RV704AZ14		G13QV1
	X		S7V1		S10V10		RV782AD14
	X		RV342A		RV738AP10		S5V1
	X		T2EM672AS		I9QW15360		G14TV1
	X		X		RV913AY14		RV783AD13
	X		T13ER2		A13ER9		S5V1
	X		V3QV5		A14ER12		G13QV1
	X		RV679AZ14		G13QV2		RV782AD14
	X		C2QV25		A14R9		S5V1
	X		A3V1		A13R12		G14TV1
	X		RV683A		A14V1D13		RV783AD13
	X		A15V1		S13W16384D13		S5V1
	X		A2EM0+		G14TV1		G13QV1
	X		A3EV1		A14R0		S5V1B14
	X		C2QV24		A13V1D14		G14TV0
	X		T2ER12		S14W16384D14		D13QV1D13
	X		T14ER2		T13ER9		T13ER11
	X		T12R1		T14ER12		C14TV8
	X		S1V10		A8V1P8		K5TV23
F			RV702AN1		RV704A		A14R13
V3QV5			S1V1		A8EVON8		T11ER4
T13ROY14			RV691AZ1		A5EV62		T14ER5
COTV25Y14			RV404AP1		A13ER9		V0QV17
A3V1Y14			V3QV5		RV744AY13		RV837AY14
RV646AY14			RV696AZ14		T12ER13		V3QV5
A15V1			C2QV25		S5V31		RV797AZ14
A(EM)0+			A3V1		A11EV1		C2QV25
A3EV1			RV700A		I13QW16256		A3V1
COTV24			A15V1		RV763AY14		RV801A
A14ER0			A2EM0+		I13QW127		A15V1
S14V10			C2QV24		RV761AY14		A2EM0+
S7V1			A3EV1		I13QV126976		C2QV24
RV342AP14							

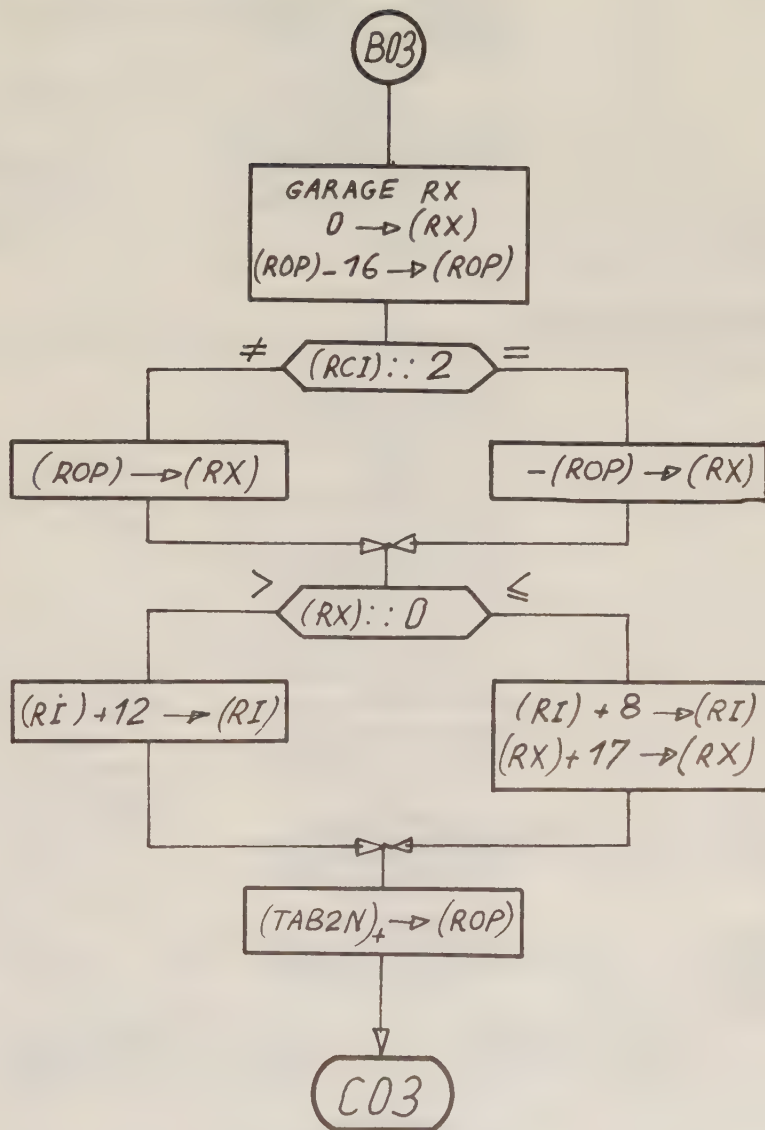
<u>800</u>	<u>850</u>	<u>900</u>
A3EV1	ML4V6D8	X
T2ER13	QL4V6B8	X
T14R2	I8QV127	X
V13QV11	S14V77	X
S11EV1Z14	RV403AP14	X
RV792AZ14	I8QV120	X
V3QV5	RV863AZ14	X
RV811AZ14	C14TV2	X
C2QV25	A15ER14	X
A3V1	A6EM893Δ+	X
RV815Δ	A7EM894Δ+	X
A15V1	ML4V6D8	X
A2EM0+	QL4V6B8	X
C2QV24	A15EM843Δ	S8V1N8
A3EV1	A6ER5	V3QV5
T2ER10	C6TV23	RV919AZ14
T14ER2	S9ER4Z1	C2QV25
T10R0	A9ER4N1	A3V1
S10V10	S7ER13Z1	RV923Δ
RV835AP10	T7ER13Z1	A15V1
K13QV3	S9V1N13	A2EM0+
A0R14	A13W16384N13	C2QV24
A0R13	A6V1D4	A3EV1
A13R0	D9TV1D9	T2R0
V3QV5	C13QV8	T2ER10
RV829AZ14	K6QV23	T14ER2
C2QV25	A13R14	S10V10
A3V1	T9ER7	RV913AN10
RV833Δ	T13ER9	A8V1N8
A15V1	A13ER2	V10QV2
A2EM0+	RV946Δ	RV739AY14
C2TV24	X	V3QV5
A3EV1	X	RV936AZ14
T2ER0	X	C2QV25
T14ER2	X	A3V1
S8R13Z11	X	RV940Δ
A8R13N11	X	A15V1
RV864AZ8	X	A2EM0+
RV842AP8	X	A3EV1
S13ER8	X	C2QV24
T13ER8	X	T2R0
A8W16384	X	T2ER10
T15EM843ΔS	X	T14ER2
X	X	S10V10
I8QV7	X	RV931AN10
RV852AZ14	X	RV739Δ
K14TV1	X	A2EM672Δ
A15ER13	X	S5EV1
A6EM879Δ+	X	RV225Δ
A7EM880Δ+	X	F
		FE

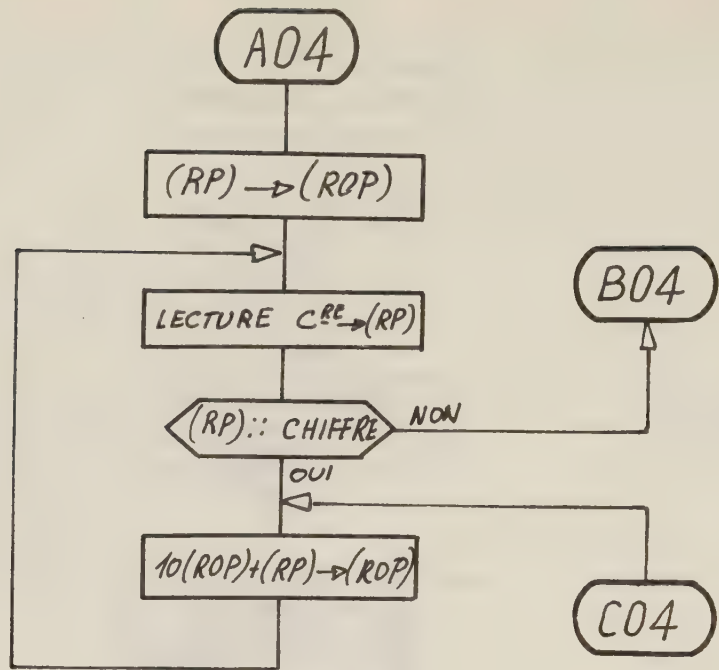




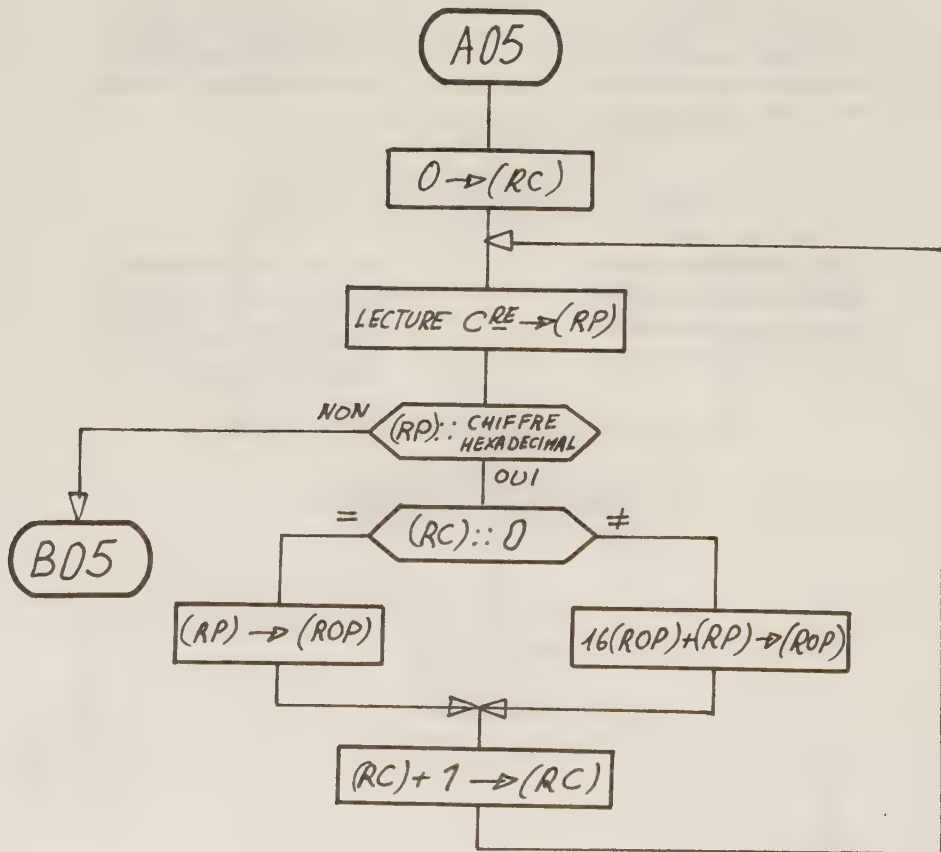
(LECTURE NOMBRE de 2 CHIFFRES DECIMAUX . CONVERSION EN BINAIRE)



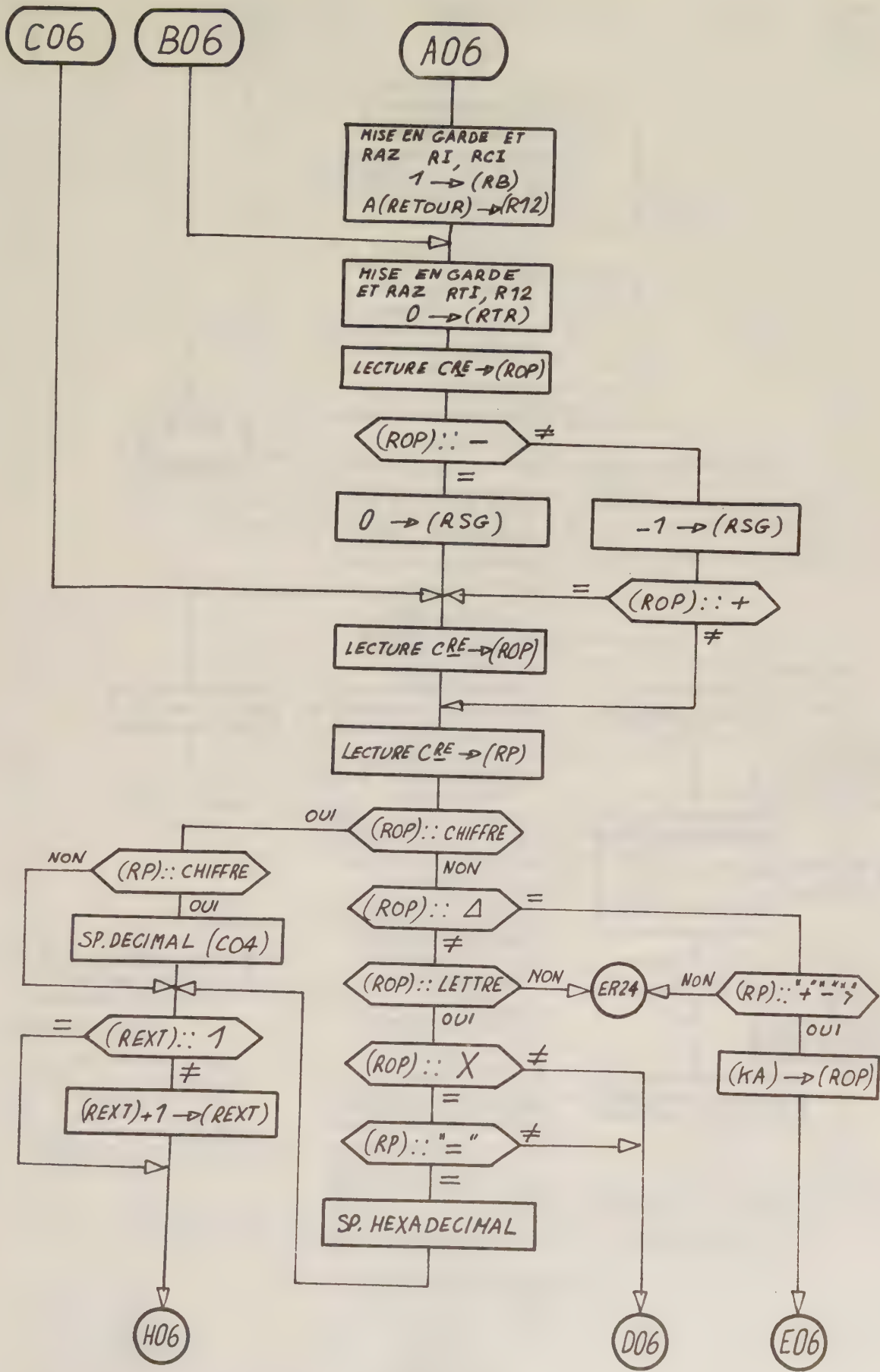


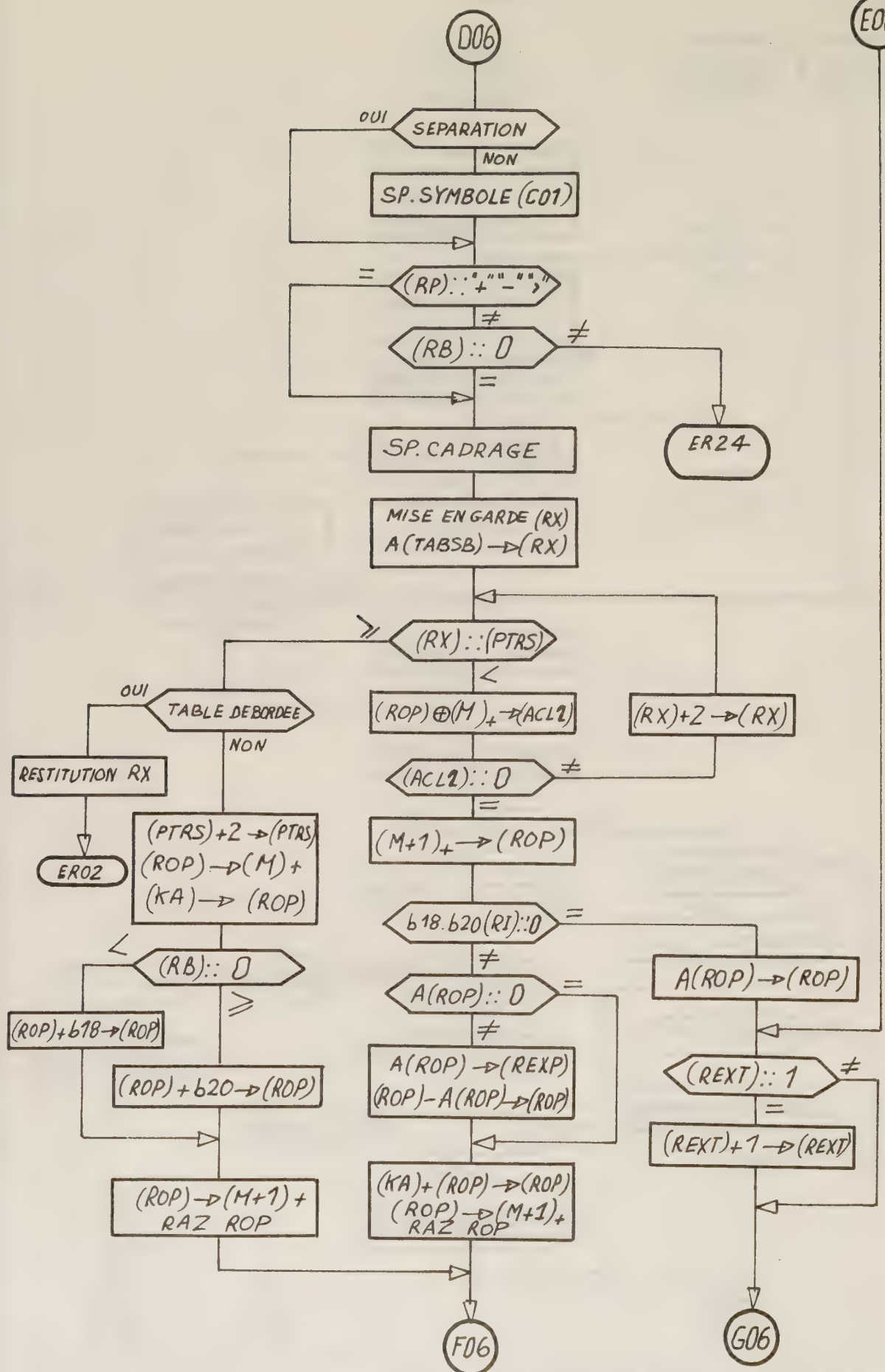


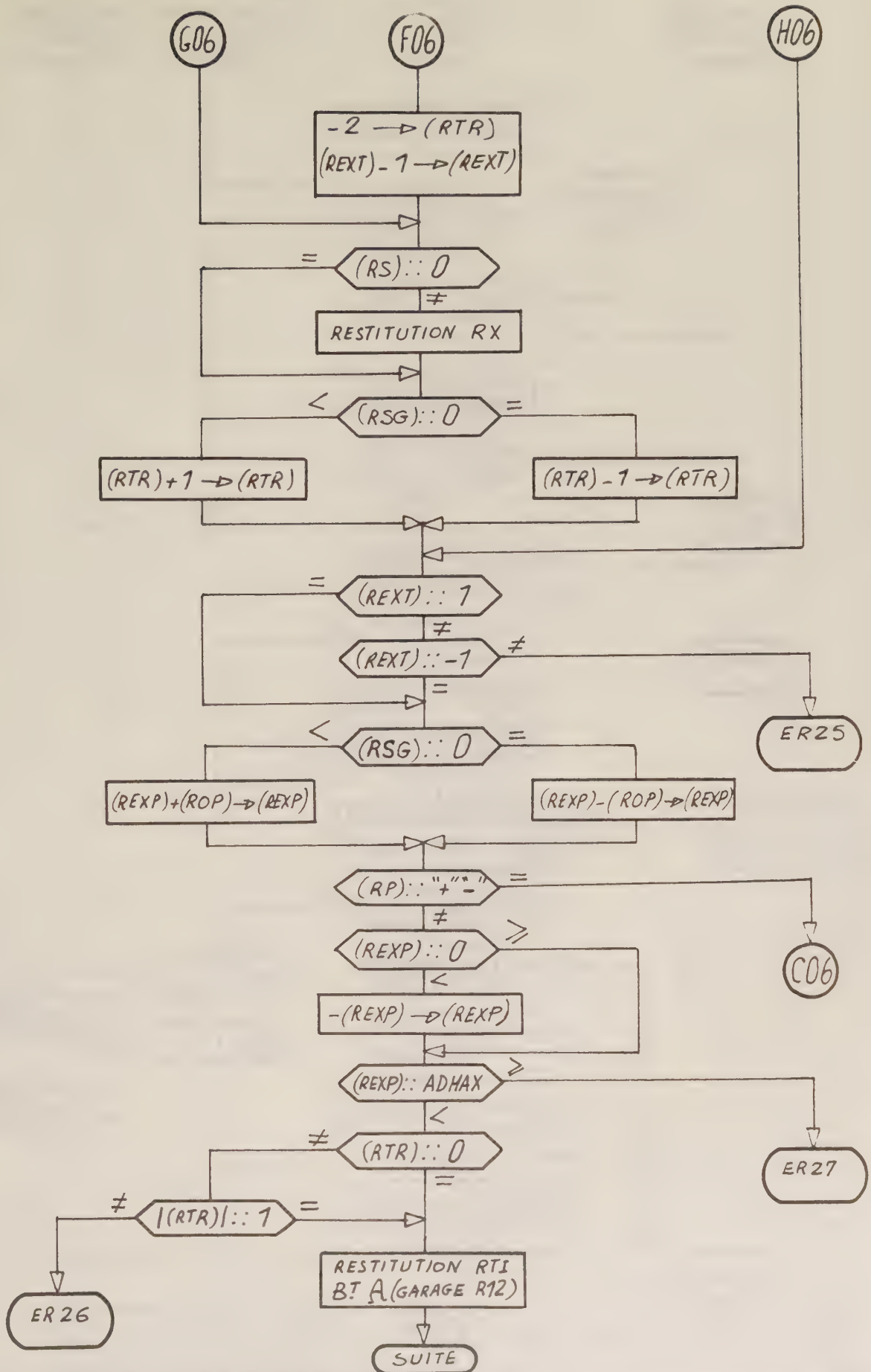
SP4 - NOMBRES DECIMAUX

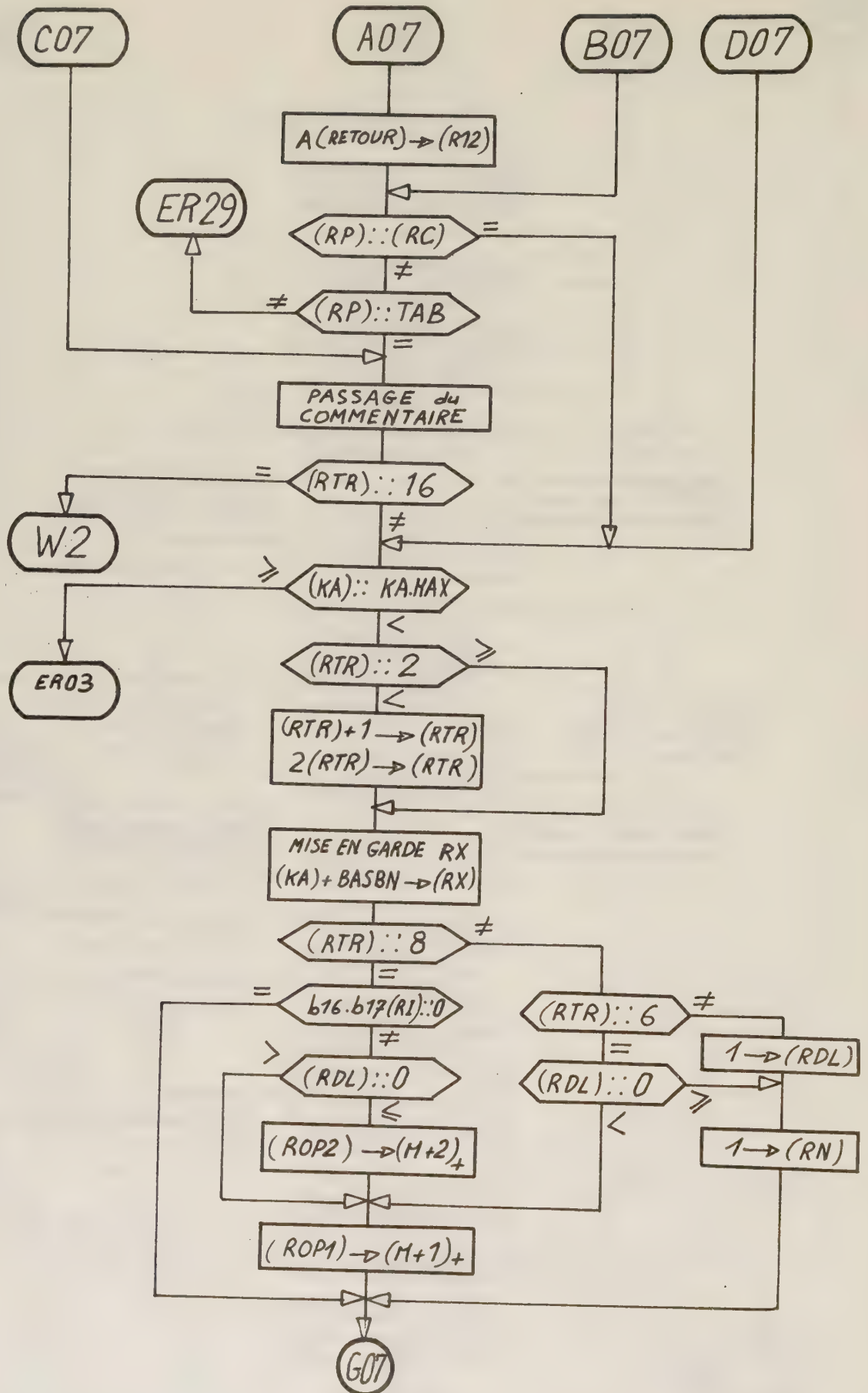


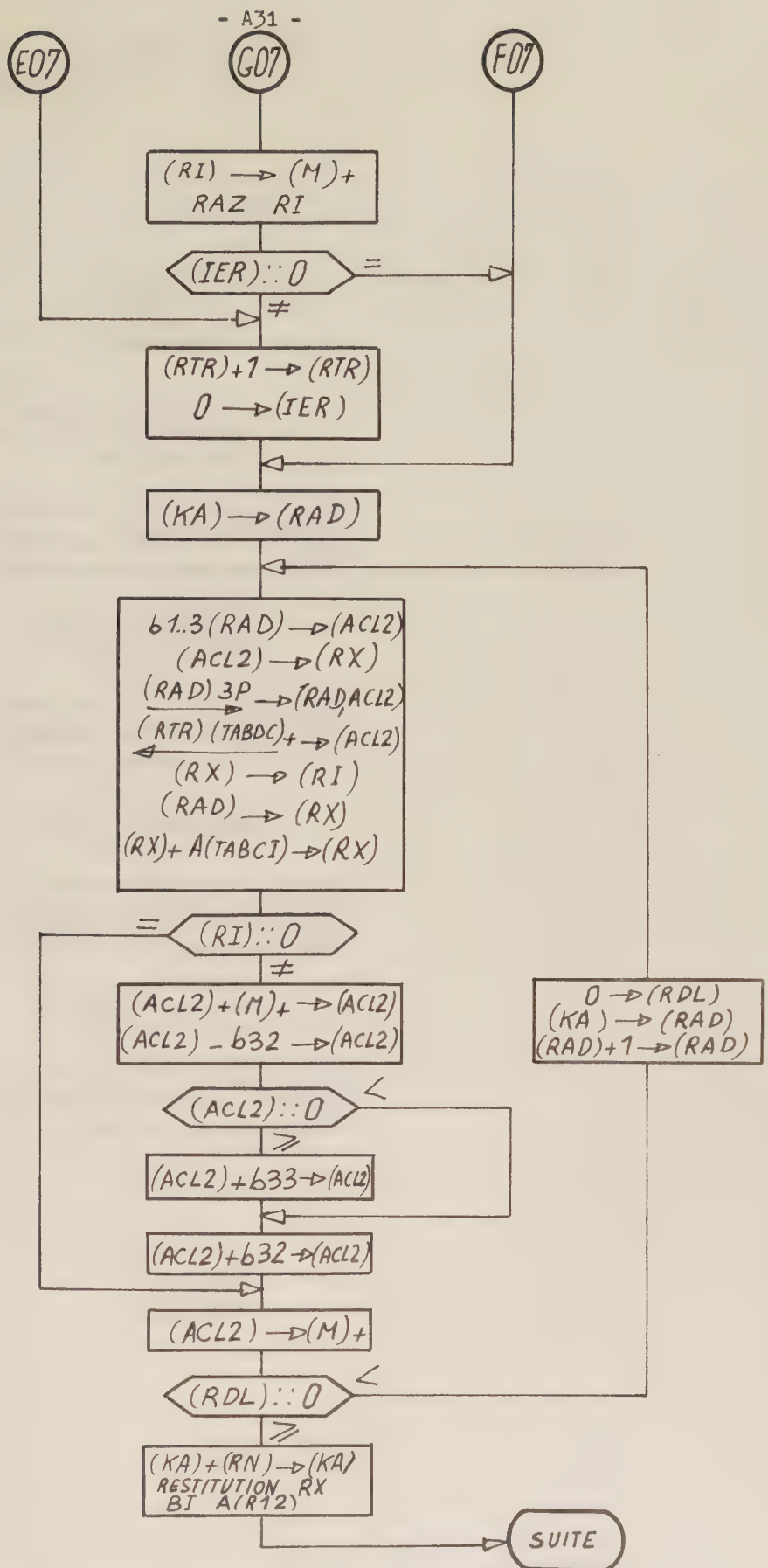
SP5 - NOMBRES HEXADECIMAUX



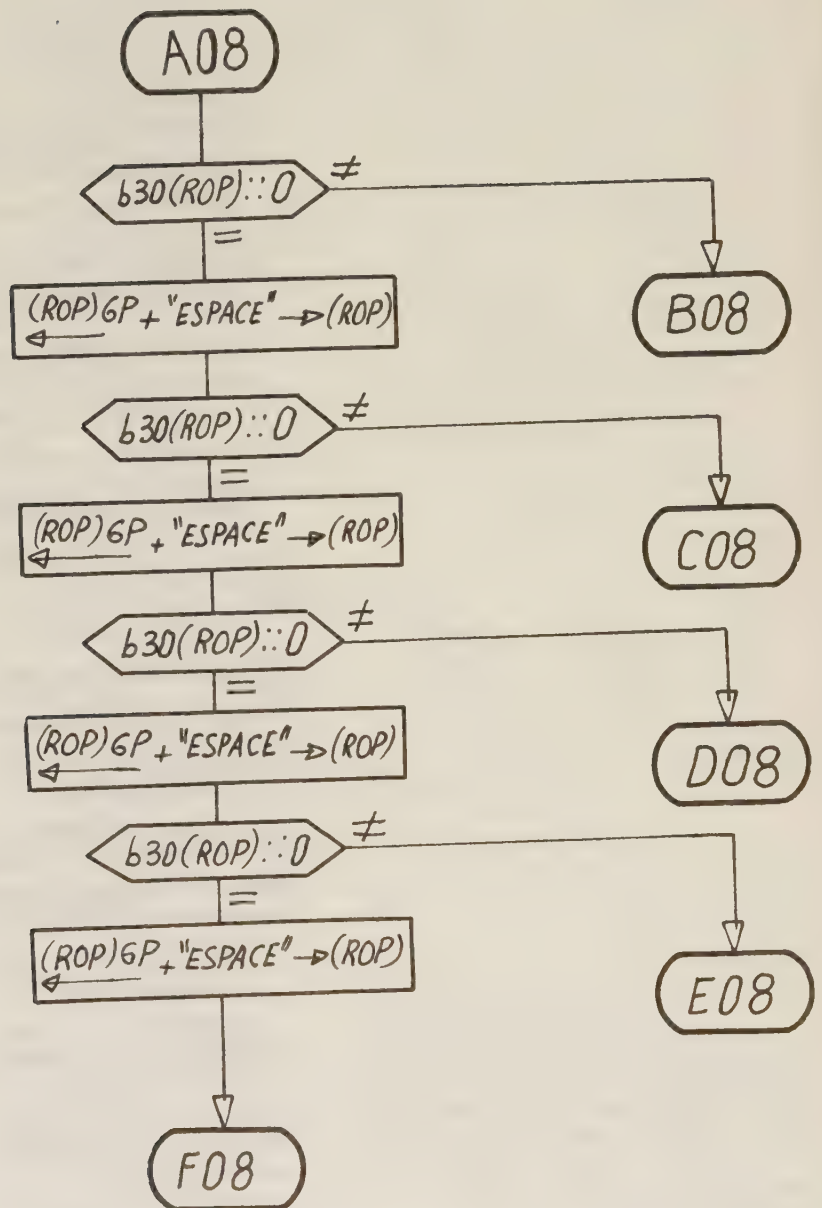








47 - SP7 TR₁ FIN d'INSTRUCTION (2)



SP ECRITURE DCB _ SP 9 _

0	K10QV19	E 2
	RV16Δ	
2	K10QV13	E 3
	RV13Δ	
4	K10QV7	E 4
	RV10Δ	
8	K10QV1	E 5
	T14R10	
	C10QV25	
	ER10	
	T14R10	
	C10QV25	
	ER10	
	T14R10	
	C10QV25	
	ER10	
	T14R10	
	C10QV25	
	ER10	
	T14R10	
	C10QV25	
	ER10	
	RR12	
	FE	

0	K10QV23	E 2
	RV39Δ	
2	K10QV15	E 4
	RV27Δ	
4	C10QV28	E 8
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	T14R10	
	C10QV27	
	S10V10	
	A10V23P10	
	A10V10	
	ER10	
	RR12	
	FE	

SP LECTURE HEXADECIMALE - SP 11.

0	A14EVO	E 2	50	A1OR14
	RV45Δ			K1OQV4
2	A14EVO	E 4		A1OEL0S
	RV31Δ			X
4	A1OEL0S	E 8		S1OV33
	X			A1OV23N10
	S1OV33			A1OV10
	A1OV23N10			A1OR14
	A1OV10			A1OW32768D10
	K1OQV4			RR12
	A1OEL0S			FE
	X			
	S1OV33			
	A1OV23N10			
	A1OV10			
	A1OR14			
	K1OQV4			
	A1OEL0S			
	X			
	S1OV33			
	A1OV23N10			
	A1OV10			
	A1OR14			
	K1OQV4			
	A1OEL0S			
	X			
	S1OV33			
	A1OV23N10			
	A1OV10			
	A1OR14			
	K1OQV4			
	A1OEL0S			
	X			
	S1OV33			
	A1OV23N10			
	A1OV10			
	A1OR14			
	K1OQV4			
	A1OEL0S			
	X			
	S1OV33			
	A1OV23N10			
	A1OV10			

EMPL.	CODE OBJET	ADRI	NO.PS	PROGRAMME SOURCE		
			1	/ PROGRAMME TEST 'ASSEMBLEUR' (TYPES D'INSTRUCTION ET OPERANDES)		
			2			
			3			
			4			
0000	98 99E00800		5	TEST1	AR	1,15
0001	98 99E00408		6	ALPHA	AI	64,15
0002	98 984007C8		7		AI	X=7C,2
0003	98 986035CC		8		AI	x860,3
0004	98 98003A1C		9		AI	xx=3A1,0
0005	98 98C00014	0001	10		A	ALPHA,6
0006	6C 6C402800		11	BETA	ANR	5,2
0007	6C 6CA00064	0006	12		AN	BETA,5
0008	BA BAA00005	0000	13		CC	TABEX,5,+
0009	AC ACA00068		14		ALR	6,5
000A	AC AC8000B4	000B	15		AL	Δ+1,4
000B	C0 C080200C		16		DTI	6,4
000C	C0 C0C02008		17		DTI	X=17,6
000D	DO D0800408		18		GTI	6,4
000E	DO D0C0040C		19		GTI	X=17,6
000F	DC DC403000		20	RAD	ARCR	6,2
0010	DC DC200144	0014	21		ARC	RAD+BETA-ALPHA,1
0011	C8 C80036DA	0006	22		BCI	BETA,6,13
0012	C8 C800B806	0017	23	ΔJOK	BC	JUMP1,0
0013	C8 C8007280		24		BCR	14,2,8
0014	F8 F8001800		25		EAR	3
0015	F8 F8000154	0015	26		EA	JOK+3
0016	C8 C800BB76	0017	27		BC	JUMP1,3,7
0017	5C 5C8070D8		28	JUMP1	DMNR	4,13,14
0018	9C 9CC0A128		29		DMLR	6,18,20
0019	1C 1D801800		30		DMER	3,12
001A	9E 9EE07008		31		ILNR	7,14
001B	1C 1C400124	0012	32		IME	JOK,2
001C	18 19800084	0008	33		T	TABEX,12
001D	9A 9A600002		34		IAR	3
001E	64 64A00000		35		LNR	5
001F	B8 B9900204	0020	36		S	Δ+1,12,S
0020	00 00000000		37		NOP	

0021	66 66000068	0006	38		LN	BETA
0022	A4 A4000028		39		LLR	2
0023	A6 A6000138	0013	40	ΔGAMMA	IL	JUMP1-4
0024	98 984001C4	001C	41		A	TABEX,2
0025	C8 C8000106	0000	42		BC	ARC,1
			43		FIN	

SYMBOLS

ALPHA 0001
BETA 0006
JUMP1 0017
RAD 000F
TEST1 0000

DEFINITIONS EXTERNES

GAMMA 0023
JOK 0012

REFERENCES EXTERNES

ARC 0025
TABEX 0024

DIAGNOSTIC 0
ORGANE DE PERFORATION?
OK

PROGRAMME -TEST3- ASSEMBLE PAR CAB 502B OCT 63 3

22-10-69

1

EMPL.	CODE OBJET	ADR1	NO. PS	PROGRAMME SOURCE		
			1	/ PROGRAMME TEST 'ASSEMBLEUR ' (DIRECTIVES)		
			2			
			3			
			4			
			5			
0000			6	TEST3	RES	3
0003			7	BETA	RES	45
0030	0000000C		8		DFC	F=12
0031	40000000		9		DFC	F=0,5/31
0032	000000D8		10		DFC	F=6,75/5
0033	0000A5B0		11		DFC	X=A5B0
0034	001FC4E0		12		DFC	X=FE27000/-7
0035	00000000		13		DFC	12F=0
0041	50000004		14		DFC	S=10
0042	4E20000D		15		DFC	5S=0,5Δ4
0047	50000000		16		DFC	D=10
0048	02000000					
0049	48000000		17	F3	DFC	3D=0,6Δ3
004A	05000000					
004F	78F5C28F		18		DFC	10D=378Δ-2
0050	012E147B					
0063	34973D0F		19	GAMMA	TEXTE	TEST DIRECTIVE
0064	24A72963					
0065	34A7694F					
0066	33D2FC0F		20		TEXTE	STOP
			21		FIN	

2

SYMBOLS

BETA 0003
F3 0049
GAMMA 0063
TEST3 0000

DIAGNOSTIC 0
ORGANE DE PERFORATION?
OK

EMPL.	CODE OBJET	ADRI	NO.PS	PROGRAMME SOURCE		
			1	/ PROGRAMME TEST 'ASSEMBLEUR' (ERREURS)		
			2			
			3			
	0000		4	TEST4	RES	3
	0003		5		AR	1,2
ER 11	0004		6	BETA36	SR	3,14
	0005	0000	7		A	TEST4,1
ER 11	0006		8	ALPH=	SR	3,4
	0007		9		NOP	
ER 11	0008		10	LAX	DTI	5,12
	0009		11	BETA	ANR	5,2
	000A	0000	12		AN	TABEX,3
ER 12			13	TABEX	ARC2	6,7
			14	/		
	000C	00 00000000	15		NOP	
		ERREUR 21	16		LAI	64,2
			17	/		
	000E	00 00000000	18		NOP	
		ERREUR 22	19		OETR	30,6
	0010	00 00000000	20		NOP	
		ERREUR 22	21		DTR	13,1
	0012	00 00000000	22		NOP	
		ERREUR 23	23		QQI	34,7
	0014	00 00000000	24		NOP	
ER 13		ERREUR 24	25	BETA	A	ALPHA,3
	0016	00 00000000	26		NOP	
		ERREUR 25	27		S	TEST4+COMPT,1
			28	/		
	0018	00 00000000	29		NOP	
		ERREUR 26	30		CC	TEST4+BETA+5,7
	001A	00 00000000	31		NOP	
		ERREUR 27	32		AN	BETA+40285,13
	001C	00 00000000	33		NOP	
		ERREUR 28	34		AR	13,18
	001E	00 00000000	35		NOP	
		ERREUR 29	36		BCI	A1,5,8+
						FIN D'INSTRUCTION ILLEGALE
			37	/		
	0020	00 00000000	38		NOP	
		ERREUR 29	39		AN	BETA,6,+S
	0022	00 00000000	40		NOP	
ER 10			41	LABEL	FIN	
						ETIQUETTE INTERDITE

SYMBOLES

BETA 0009 MULTIDEF.
LABEL 0023
TEST4 0000

REFERENCES EXTERNES

A1 001F
COMPT 0017
TABEX 000A MULTIDEF.

DIAGNOSTIC 16

PROGRAMME -TEST5- ASSEMBLE PAR CAB 502B OCT 63 3

22-10-69

1

EMPL.	CODE OBJET	ADRI	NO. PS	PROGRAMME SOURCE		
			1	/ PROGRAMME TEST 'ASSEMBLEUR ' (DIRECTIVES)		
			2			
			3			
			4			
			5			
0000			6	TEST5	RES	3
0003			7	ΔARC	RES	45
0030	0000000C		8		DFC	F=12
0031	40000000		9		DFC	F=0,5/31
0032	000000D8		10		DFC	F=6,75/5
0033	0000A5B0		11		DFC	X=A5B0
0034	001FC4E0		12		DFC	X=FE27000/-7
0035	00000000		13		DFC	12F=0
0041	50000004		14		DFC	S=10
0042	4E20000D		15		DFC	5S=0,5Δ4
0047	50000000		16		DFC	D=10
0048	02000000					
0049	4B000000		17	ΔTABEX	DFC	3D=0,6Δ3
004A	05000000					
004F	78F5C28F		18		DFC	10D=378Δ-2
0050	012E147B					
0063	34973D0F		19	TEXTE	TEST DIRECTIVE	
0064	24A72963					
0065	34A7694F					
0066	33D2FC0F		20	TEXTE	STOP	
0067	48 48000366	0000	21	DEBUT	BRC	JOK, 3,6
0068	08 08000006	0000	22		BC	GAMMA, 0
0067			23		FIN	DEBUT

2

SYMOLES

DEBUT 0067
TEST5 0000

DEFINITIONS EXTERNES

ARC 0003
TABEX 0049

REFERENCES EXTERNES

GAMMA 0068
JOK 0067

DIAGNOSTIC 0
ORGANE DE PERFORATION?
OK

PROGRAMME -TEST6- ASSEMBLE PAR CAB 502B OCT 63 3

28-10-69

1

EMPL.	CODE OBJET	ADRI	NO.PS	PROGRAMME SOURCE		
			1	/ PROGRAMME TEST 'ASSEMBLEUR ' (DIRECTIVES)		
			2			
			3			
			4			
			5			
0000			6	TEST6	RES	3
0003			7	ΔARC	RES	45
0030	0000000C		8		DFC	F=12
0031	40000000		9		DFC	F=0,5/31
0032	000000D8		10		DFC	F=6,75/5
0033	0000A5B0		11		DFC	X=A5B0
0034	001FC4E0		12		DFC	X=FE27000/-7
0035	00000000		13	ΔCOMPT	DFC	12F=0
0041	50000004		14		DFC	S=10
0042	4E20000D		15		DFC	5S=0,5Δ4
0047	50000000		16		DFC	D=10
0048	02000000					
0049	4B000000		17	ΔTABEX	DFC	3D=0,6Δ3
004A	05000000					
004F	78F5C28F		18		DFC	10D=378Δ-2
0050	012E147B					
0063	34973D0F		19	TEXTE	TEST DIRECTIVE	
0064	24A72963					
0065	34A7694F					
0066	33D2FC0F		20	TEXTE	STOP	
0067	48 48000366	0000	21	DEBUT	BRC	JOK, 3,6
0068	C8 C8000006	0000	22		BC	GAMMA, 0
			23		FIN	

2

SYMBLES

DEBUT 0067
TEST6 0000

DEFINITIONS EXTERNES

ARC 0003
COMPT 0035
TABEX 0049

REFERENCES EXTERNES

GAMMA 0068
JOK 0067

DIAGNOSTIC 0
ORGANE DE PERFORATION?
OK

D

ETAT DE CHARGEMENT PAR CAB 502B OCT 63

28-10-69

1

NO.PR	NOM	DEBUT	FIN
1	TEST1	0A(X)	0A25

DEFINITIONS EXTERNES

	PROGRAMME	1
GAMMA	0A23	
JOK	0A12	

REFERENCES EXTERNES

	PROGRAMME	1	
ARC	0A25		* NONDEF. *
TABEX	0A24		* NONDEF. *

DIAGNOSTIC	2	NON EXECUTION
------------	---	---------------

D

ETAT DE CHARGEMENT PAR CAB 502B OCT 63

28-10-69

1

NO. PR	NOM	DEBUT	FIN
1	TEST1	0A00	0A25
2	TEST6	0A26	0A8E

DEFINITIONS EXTERNES

	PROGRAMME	1
GAMMA	0A23	
JOK	0A12	

	PROGRAMME	2
ARC	0A29	
COMPT	0A5B	
TABEX	0A6F	

REFERENCES EXTERNES

	PROGRAMME	1
ARC	0A25	0A29
TABEX	0A24	0A6F

	PROGRAMME	2
GAMMA	0A8E	0A23
JOK	0A8D	0A12

DIAGNOSTIC 0
PAS D'ADRESSE D'EXECUTION

D

ETAT DE CHARGEMENT PAR CAB 502B OCT 63

28-10-69

1

NO.PR	NOM	DEBUT	FIN
1	TEST1	0A00	0A25
2	TEST6	0A26	0A8E
3	TEST5	0A8F	0AF7

DEFINITIONS EXTERNES

	PROGRAMME	1	
GAMMA	0A23		
JOK	0A12		
	PROGRAMME	2	
ARC	0A29	* MULTIDEF. *	0A92
COMPT	0A5B		
TABEX	0A6F	* MULTIDEF. *	0AD8
	PROGRAMME	3	
ARC	0A92	MULTIDEF.	
TABEX	0AD8	MULTIDEF.	

REFERENCES EXTERNES

	PROGRAMME	1	
ARC	0A25		0A29
TABEX	0A24		0A6F
	PROGRAMME	2	
GAMMA	0A8E		0A23
JOK	0A8D		0A12
	PROGRAMME	3	
GAMMA	0AF7		0A23
JOK	0AF6		0A12
DIAGNOSTIC	2	NON EXECUTION	

D

ETAT DE CHARGEMENT PAR CAB 502B OCT 63

28-10-69

1

NO. PR	NOM	DEBUT	FIN
1	TEST1	0A00	0A25
2	TEST2	0A26	0A69
3	TEST3	0A6A	0AD0

DEFINITIONS EXTERNES

	PROGRAMME	1
GAMMA	0A23	
JOK	0A12	

REFERENCES EXTERNES

	PROGRAMME	1
ARC	0A25	* NONDEF. *
TABEX	0A24	* NONDEF. *
DIAGNOSTIC	2	NON EXECUTION

D

ETAT DE CHARGEMENT PAR CAB 502B OCT 63

29-4-1970 1

NO.PR	NOM	DEBUT	FIN
1	TEST5	0A00	0A68
2	TEST1	0A69	0A8E

DEFINITIONS EXTERNES

	PROGRAMME	1
ARC	0A03	
TABEX	0A49	

	PROGRAMME	2
GAMMA	0A8C	
JOK	0A7B	

REFERENCES EXTERNES

	PROGRAMME	1
GAMMA	0A68	0A8C
JOK	0A67	0A7B

	PROGRAMME	2
ARC	0A8E	0A03
TABEX	0A8D	0A49

DIAGNOSTIC 0

ADRESSE D'EXECUTION 0A67 PR1

